Open Service Interface Definition

# Assessment

Document Release: 2.0
OSID version 2.0

## Summary

The Assessment Open Service Interface Definition (OSID) supports creating, organizing, administrating, evaluating, storing and retrieving Assessments. Assessments are organized into Sections and within Sections into Items. Once composed, an Assessment can be Published in which case it is read to be Taken by an Agent[1] (Student). Each Assessment, Section, or Item taken can have an Evaluation.

Assessment, Section, and Item include the following:

- Id
- Type
- DisplayNames
- Description
- Properties set by the implementation

The AssessmentManager keeps track of Assessments as well as returning the Types that are supported by a particular implementation.

There may be relationships between this OSID and others and at different levels. For example, an implementation might manage an Assessment's content, question banks, etc as a digital repository using the Repository OSID. A PublishedAssessment can include an optional reference both to a CourseSection, an element of the coursemanagement OSID, and to a GradableObject, an element of the grading OSID. The grading OSID might be appropriate for summative Evaluations of an Assessment.

[1] Refer to documentation on Agent OSID for more information on Agents.

## Service Definition

An examination of an Open Service Interface Definition (OSID) usually begins with the Manager. All Managers[2] provide the way to create the objects that implement the principal interfaces in the Service. Before looking at the Manager's methods, it may be helpful to examine the composition of Assessments as well as the Assessment processes at a high level.

Assessments run the range from the extremely simple, such as a single question, to the highly complex, such as a multi-part, hierarchical, structure that is unique both in order and content each time it is taken.

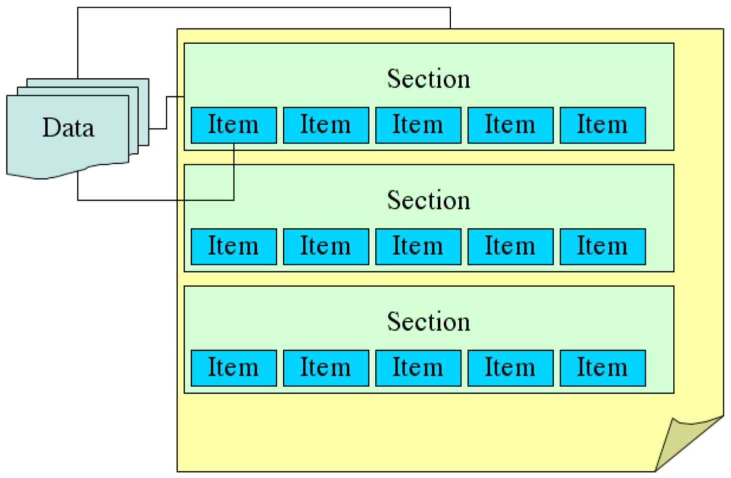

**Figure 1: Structure of an Assessment**

Whatever the scope, Assessments contain both one or more Sections as well as any data about the Assessment that may be useful. Examples of such Assessment data are:

- objectives of the Assessment
- relevant links
- current and historical usage data
- controlled release information
- submission rules
- scoring and critique rules
- notification rules
- tracking of students actions

Sections are ordered within the Assessment, although as we will see that order is flexible, and may contain other Sections. Sections also contain one or more Items as well as any useful Section-specific data such as instructions or media.

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

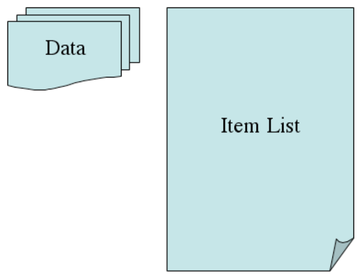

**Figure 2: Structure of a Section**

As with Sections in an Assessment, Sections within a Section and Items with a Section have an order. Items are the basic unit of Assessment and they too can contain Item data such as questions, possible answers, instructions, media, and so on.

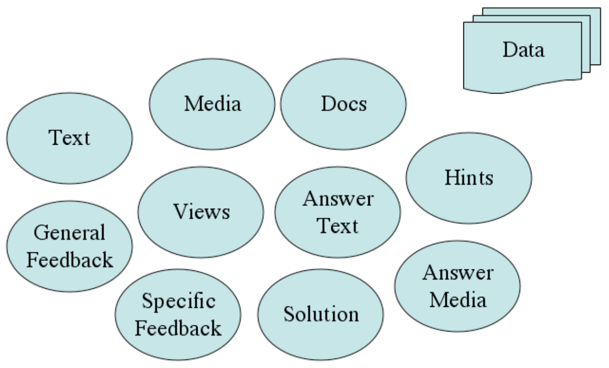

**Figure 3: Structure of an Item**

The order of Sections within an Assessment and Items within a Section plays an important pedagogical role. Initially Sections and Items are will be returned in the order, first-in, first-out (FIFO) that they were added. There are methods to change the order in which Sections and Items are returned. One might want to change the order such that any particular Student would not necessarily have the same question order as the next Student.

Applications may exist for authoring Assessments to match this OSID. Such an application might look like this, although the specifics would vary:

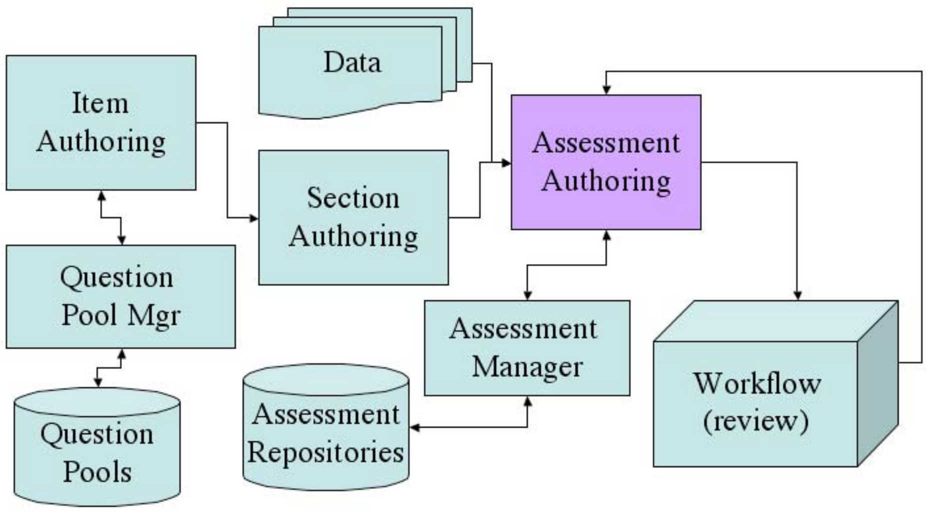

**Figure 4: Possible Assessment Authoring System**

With an overview of how Assessments are organized, let us next look at the Assessment process. The first step is to create an Assessment. The Assessment can contain all new material, a new organization of pre-existing Sections or Items, or a combination of the two. After it is created, the Assessment is published or released.

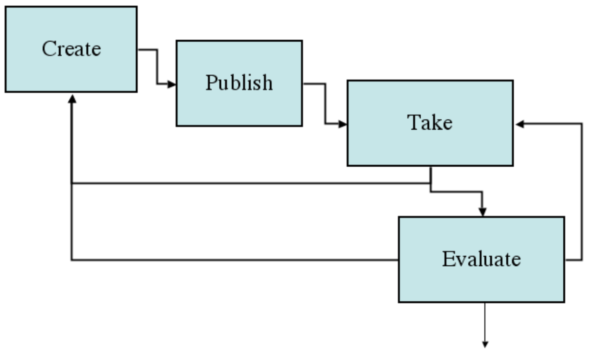

**Figure 5: The Assessment Process**

Publication is the point at which an Assessment in general, is made specific to a context such as being for a particular audience or available at a specific time. Once published, the Assessment is taken. This is the interaction of the Agent (usually a Student) with the Assessment and its Sections and Items. During or after this step, the Assessment, Section, and Items can be evaluated. In a dynamic or adaptive mode, what is created, taken, or both can be determined by the result of the evaluation or some other rule.

## org.osid.assessment.AssessmentManager

The AssessmentManager is primarily responsible for the creation and retrieval of Assessments, Sections, and Items both before and after they are taken. The **createAssessment**() method creates an Assessment.[3] The arguments to the method are a name for display, a description, and an Assessment Type.[4]

[3] Refer to the discussion of OsidManager for information on create methods and their use in implementation substitution.

The Assessment Type is one of several Types in this OSID and Types are used widely throughout the OSIDs. A Type is a way of characterizing something and makes use of 4 properties: an authority, domain, keyword, and description. Sample AssessmentTypes might include:

- Multi-Part
- Single-Part
- Standardized Oral Proficiency (SOPI)
- Document Submission
- Secure Test
- Homework

The **createAssessment**() method's implementation will assign a unique identifier[5] to the Assessment. An identifier is unambiguous while a name may not be. The AssessmentManager is responsible for storing this new Assessment and as well as any other created objects. Any Assessment created in this fashion can be removed using the Manager's **deleteAssessment**() method. The **getAssessments**() method returns the set of Assessments being managed.[6] A variation on this method returns all the Assessments of a particular AssessmentType. The **getAssessment**() method returns a specific Assessment.

It is important to note that Sections are created and can exist independently from Assessments. The same Section can be reused in more than one Assessment. Similarly, Items are independent. This model means that the AssessmentManager is solely responsible for the lifecycle of Assessments, Sections, and Items. The Assessment, in turn, simply adds, removes, and orders its Sections and the Section does the same for its Sections and Items. The **createSection**(), **deleteSection**(), and **getSections**() methods handle creating, deleting, and enumerating Sections and the **createItem**(), **deleteItem**(), and **getItems**() methods do the same for Items. **getSection**() gets a Section by Id and **getItem**() does the same for an Item. Both Sections and Items have Types. Sample Sections Types are: static, and randomized of various kinds. Sample Items Types are:

- multiple-choice
- true-false
- short answer
- matching
- ordering
- Likert-scale rating
- essay

The specific Types used are outside the scope of the OSID and will develop within the community.

Assessments are first published and then can be taken. Within an AssessmentTaken are Sections and Items taken. The **createAssessmentPublished**() method publishes an Assessment; the **deleteAssessmentPublished**() method deletes it, the **getAssessmentPublished**() method gets a specific AssessmentPublished and the **getAssessmentsPublished**() method returns all the published Assessments. An AssessmentPublished is not created with the Manager, rather it is created by an implementation through a process outside the scope of this OSID. The AssessmentPublished includes data of any kind, examples of which are:

---

[4] Refer to "Types" document for more information on Types in OSID.

[5] Refer to the discussion of Unique Identifiers in "Ids" document.

[6] Objects are returned through an iterator, in this particular case a GradeRecordIterator. Iterators return objects in sequence. Refer to the general discussion of Iterators for more information.

- release date and time
- submission date and time
- release rules
- passwords for proctored release
- IP mask
- availability of feedback, solutions, and scores
- submission rules
- scoring rules
- notification rules

## AssessmentManager Method Summary

| | |
|---|---|
| Assessment | createAssessment(String name, String description, Type assessmentType) |
| AssessmentPublished | createAssessmentPublished(Assessment assessment) |
| Item | createItem(String name, String description, Type itemType) |
| Section | createSection(String name, String description, Type sectionType) |
| void | deleteAssessment(Id AssessmentId) |
| void | deleteAssessmentPublished(Id AssessmentPublishedId) |
| void | deleteItem(Id itemId) |
| void | deleteSection(Id sectionId) |
| Assessment | getAssessment(Id assessmentId) |
| AssessmentPublished | getAssessmentPublished(Id assessmentPublishedId) |
| AssessmentIterator | getAssessments() |
| AssessmentIterator | getAssessmentsByType(Type assessmentType) |
| AssessmentPublishedIterator | getAssessmentsPublished() |
| TypeIterator | getAssessmentTypes() |
| TypeIterator | getEvaluationTypes() |
| Item | getItem(Id itemId) |
| ItemIterator | getItems() |
| ItemIterator | GetItemsByType(Type itemType) |
| TypeIterator | getItemTypes() |

| | |
|---|---|
| Section | getSection(Id sectionId) |
| SectionIterator | getSections() |
| SectionIterator | getSectionsByType(Type sectionType) |
| TypeIterator | GetSectionTypes() |

## org.osid.assessment.Assessment

The Assessment is the top-level organizing structure for this Service. Assessments contain Sections which contain Items. Assessments are created by the Manager and should have general data. An AssessmentPublished is an Assessment that has additional data added to make it specific to a context. The context might be a particular date and time the Assessment is to be taken, the list of Agents who can interact with the Assessment and their roles, submission rules, etc. The same Assessment might be published several times. Assessments are created through the Manager.

The **getDisplayName**() and **getDescription**() methods return the name and description for the Assessment, respectively. These values are initialized by an implementation to values passed as arguments to the create method. These can be modified with **updateDisplayName**() and **updateDescription**(). The create method also provides a unique identifier returned by **getId**(); this value is immutable. The create method includes and argument for the AssessmentType and that value is returned by **getAssessmentType**().

Assessments add and remove the Sections they contain with **addSection**() and **removeSection**(), respectively. The contained Sections are returned by **getSections**(). The order of Sections can be changed through the **orderSections**() method. The method accepts an array of Sections and a subsequent call to **getSections**() will use the order of Sections in that array. Section ordering can be dynamic and adaptive through this mechanism.

An Assessment can contain any kind of additional data as long as it is Serializable. This data might be Section instructions, media, or other information. The **getData**() and **updateData**() methods control access to this data. The kind, format, and interpretation of this data are outside the scope of this OSID. An implementation might or might not know what to do with this data; it might just store it. This data might be meaningful to a single application or standardized across applications in an industry. One special piece of data is a Topic String associated with this Section. The **getTopic**() and **updateTopic**() methods manage this data. Topics are likely to be standardized within an industry or community.

### Assessment Method Summary

| | |
|---|---|
| void | addSection(Section section) |
| Type | getAssessmentType() |
| Serializable | getData() |
| String | getDescription() |
| String | getDisplayName() |
| Id | getId() |
| PropertiesIterator | getProperties() |

| | |
|---|---|
| Properties | getPropertiesByType() |
| TypeIterator | getPropertyTypes() |
| SectionIterator | getSections() |
| String | getTopic() |
| void | orderSections(Section[] sections) |
| void | removeSection(Id sectionId) |
| void | updateData(Serializable data) |
| void | updateDescription(String description) |
| void | updateDisplayName(String displayName) |
| void | updateTopic(String topic) |

## org.osid.assessment.AssessmentPublished

An AssessmentPublished is an Assessment plus information about the access or release of this Assessment to Agents with Roles. The kind of release-related information and who can access the information and under what circumstances and in what capacity are all folded into the Serializable data for this Assessment. An AssessmentPublished is created by the Manager. Similar to Assessment, there are methods for name, description, Id and data, specifically: **getDisplayName**(), **getDescription**(), **getId**(), **getData**(), **updateDisplayName**(), **updateDescription**(), and **updateData**(). Note that these may or may not be tied to the underlying Assessment, which is returned by **getAssessment**(). That is a choice of the implementation. An AssessmentPublished is linked to a specific CourseSection and a GradingAssignment, elements in the ClassAdmin and Grading OSIDs, respectively. The **getCourseSectionId**(), **updateCourseSectionId**(), **getGradingAssignmentId**(), and **updateGradingAssignmentId**() methods capture these external references. AssessmentsPublished can have a date associated with them and **getDate**() returns this.

An AssessmentPublished can be taken. The methods **createAssessmentTaken**(), **deleteAssessmentTaken**(), and **getAssessmentsTaken**() manage the lifecycle of an AssessmentTaken. AssessmentsTaken can have Evaluations. For each Agent (Student) that takes the Assessment, there is an AssessmentTaken and its Evaluations. Use the **getEvaluations**() method for the AssessmentPublished to get all the Evaluations for all those who took the Assessment. Use same method for the AssessmentTaken to get all the Evaluations for a particular Agent (student).

The AssessmentPublished might fit into a application for controlling the release of Assessments. Such a system might look like this, although the specifics would vary:

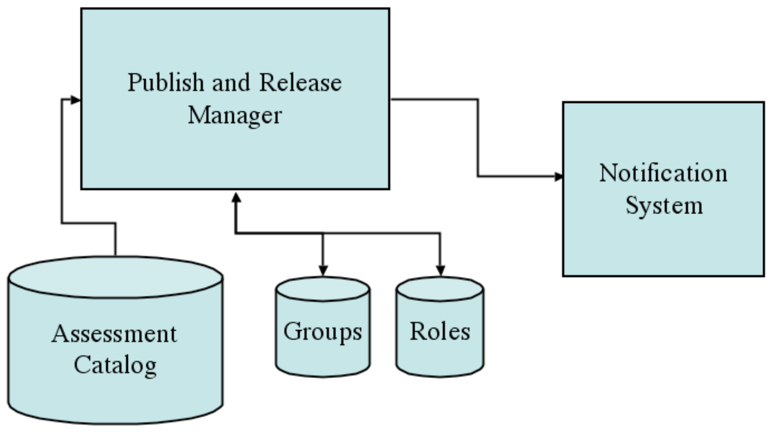

**Figure 6:  Possible Assessment Release Application**

## AssessmentPublished Method Summary

| | |
|---|---|
| AssessmentTaken | createAssessmentTaken(Id agentId) |
| void | deleteAssessmentTaken(Id assessmentTakenId) |
| Assessment | getAssessment() |
| AssessmentTakenIterator | getAssessmentsTaken() |
| AssessmentTakenIterator | getAssessmentsTakenBy(Id agentId) |
| Id | getCourseSectionId() |
| Serializable | getData() |
| long | getDate() |
| String | getDescription() |
| String | getDisplayName() |
| EvaluationIterator | getEvaluations() |
| EvaluationIterator | getEvaluationsByType(Type evaluationType) |
| Id | getGradingAssignmentId() |
| Id | getId() |
| void | updateCourseSectionId(Id id) |
| void | updateData(Serializable data) |
| void | updateDescription(String description) |
| void | updateDisplayName(String displayName) |

| | |
|---|---|
| void | updateGradingAssignmentId(Id id) |

## osid.assessment.AssessmentTaken

Once and Assessment is published, it may be taken by an Agent. What data is added to an Assessment-Taken, which presumably includes responses to questions as well as other information about the process, is added by the implementation and such monitoring is outside the scope of the OSID. An application that controls taking an Assessment might include steps for accessing, previewing, starting, ending, and post-processing the Assessment. Between the start and end, the process of monitoring an adaptive Assessment might look something like this:

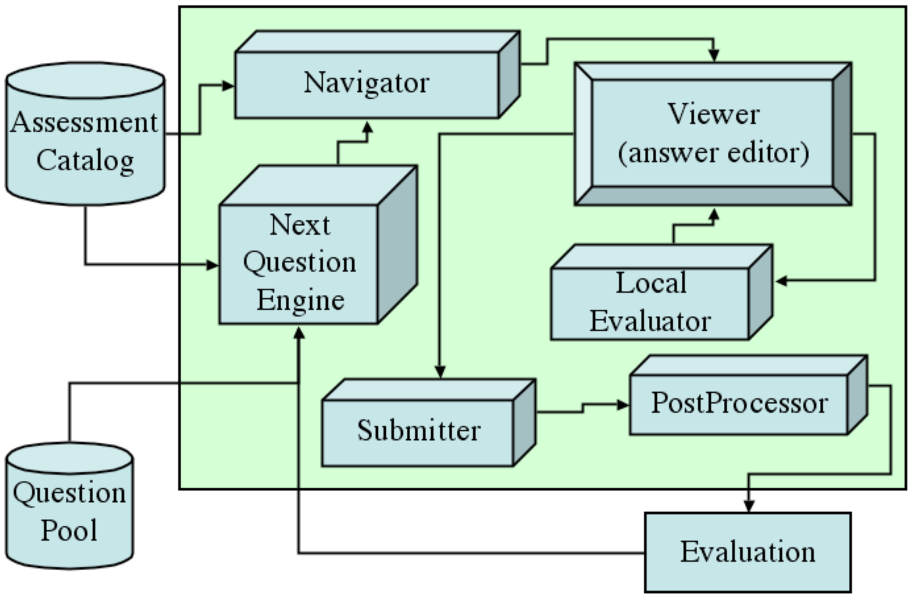

**Figure 7: Possible Assessment Monitoring Application**

This design might use the Repository OSID to manage an Assessment Catalog and Question Pool. The Assessment OSID implementation might also include the monitoring and control of how an Assessment is taken and evaluated. Systems will vary as to how an AssessmentTaken is derived.

The AssessmentTaken is based on an AssessmentPublished and the **getAssessmentPublished**() method returns that. A specific Agent (Student) took the AssessmentPublished and the **getAgentId**() method returns that. There is support for a single timestamp for when the Assessment was taken and the **getDate**() method returns that. Naturally, an implementation might well capture and provide much finer-grained timing information, down to the Item level. That and other data can be accessed and stored through the **getData**() and **updateData**() methods. The **createSectionTaken**() and **deleteSection-Taken**() methods create and delete SectionsTaken, respectively. The list of SectionsTaken is returned by the **getSectionsTaken**() method. While the order of Sections in the Assessment is variable, the order of Sections actually taken is in the fixed order they were taken. An Assessment can have zero or more Evaluations of various EvaluationTypes. The **createEvaluation**(), **deleteEvaluation**(), and **getEvalua-tions**() methods manage this. These Evaluations also can be returned in aggregate for all Assessment-sTaken by calling the **getEvaluations**() method on the AssessmentPublished. A similar approach will be used the SectionsTaken and ItemsTaken described later on.

### AssessmentTaken Method Summary

| | |
|---|---|
| Evaluation | createEvaluation(Type evaluationType) |

| | |
|---|---|
| SectionTaken | createSectionTaken(Section section) |
| void | deleteSectionTaken(Id sectionTakenId) |
| Id | getAgentId() |
| AssessmentPublished | getAssessmentPublished() |
| Serializable | getData() |
| long | getDate() |
| String | getDisplayName |
| Id | getId() |
| EvaluationIterator | getEvaluations() |
| EvaluationIterator | getEvaluationsByType(Type evaluationType) |
| SectionTakenIterator | getSectionsTaken() |
| void | deleteEvaluation(Id evaluationId) |
| void | updateData(Serializable data) |
| void | updateDisplayName(String displayName) |

## org.osid.assessment.Evaluation

An AssessmentTaken, SectionTaken, or ItemTaken may have an Evaluation. An Evaluation can range from the simple, such as correct / incorrect, to a complex analysis with feedback and media. Evaluation can take place while an Agent is interacting with an Item (a local Evaluation for immediate feedback) or after an Item or Section has been submitted (for adaptive ordering or questions generation, or for interim results). Evaluation can wait until the entire Assessment has been completed. A single Agent can create an Evaluation or a complex system can distribute that responsibility to several Agents, perhaps each evaluating a single Item or Section or a random allocation. Evaluations might also be part of workflow for review or the work of several Agents (Students) might be treated as a group effort with a common Evaluation. Here is an example of a complex system for authoring, distributing, and viewing Evaluations:

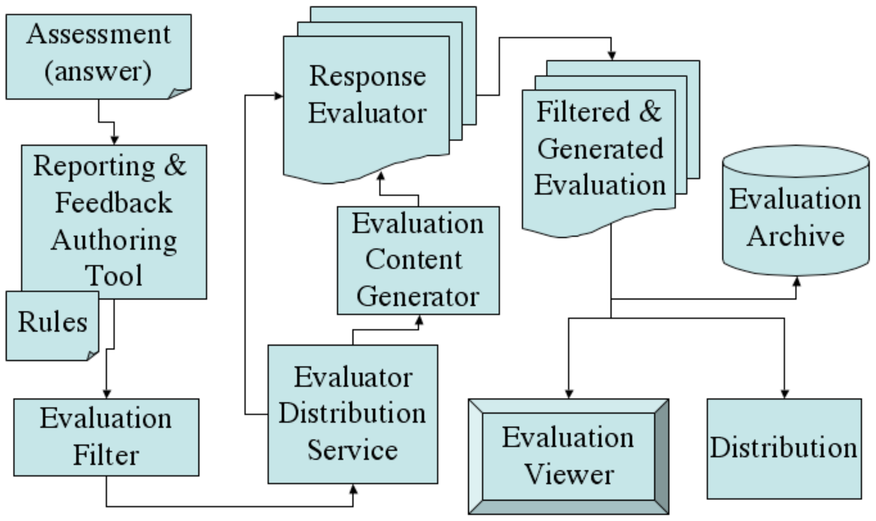
Assessment (answer)
Reporting & Feedback Authoring Tool
Rules
Evaluation Filter
Response Evaluator
Evaluation Content Generator
Evaluator Distribution Service
Filtered & Generated Evaluation
Evaluation Archive
Evaluation Viewer
Distribution

**Figure 8: Possible Complex Evaluation Application**

With knowledge of this kind of variability, the Evaluation needs to be very extensive or very general. The OSID chooses a general model. Each Evaluation is created through the Manager and has an EvaluationType. The **getType**() method returns that Type. The **getId**() method returns a unique identifier for the Evaluation. The **getModifiedBy**() and **getModifiedDate**() methods identify which Agent made the Evaluation and when, respectively. The **getObjectTaken**() method returns the thing being Evaluated and the **getData**() and the **updateData**() methods return the Evaluation itself – the specifics of which are not specified by the OSID.

## Evaluation Method Summary

| | |
|---|---|
| Serializable | getData() |
| String | getDisplayName() |
| Id | getId() |
| Id | getModifiedBy() |
| long | getModifiedDate() |
| Id | getObejctTaken() |
| Type | getType() |
| void | updateData(Serializable data) |
| void | updateDisplayName(String displayName) |

## org.osid.assessment.Item

The Item is the basic unit of Assessment. The Item may be a simple, self-describing question or a complex interactive exercise with instructions, media, reading material, multiple responses, and so on. Again, the OSID does not define this, but provides for a general data object to hold whatever is needed. The **getData**() and **updateData**() methods handle access to and setting this data. Items are created through the Manager with a create method that accepts a name, description, and ItemType. The create method's implementation will provide the unique Id for the Item, set the initial values for the DisplayName and Description, and set the ItemType. The **getDisplayName**(), **getDescription**(), **updateDisplayName**(), **updateDescription**(), **getId**(), and **getItemType**() methods play the same roles as elsewhere accessing and setting these properties.

### Item Method Summary

| | |
|---|---|
| Serializable | getData() |
| String | getDescription() |
| String | getDisplayName() |
| Id | getId() |
| Type | getItemType() |
| PropertiesIterator | getProperties() |
| Properties | getPropertiesByType(Type propertiesType) |
| TypeIterator | getPropertyTypes() |
| void | updateData(Serializable data) |
| void | updateDescription(String description) |
| void | updateDisplayName(String displayName) |

## osid.assessment.ItemTaken

The ItemTaken is an Item, any Evaluations, and whatever data reflects the being taken. Such data would include the response and possibly much more depending on the application. The **getItem**() method returns the Item that was taken. The **createEvaluation**(), **deleteEvaluation**(), and **getEvaluations**() methods manage the Evaluations for this ItemTaken. The **getSectionTaken**() method returns the SectionTaken for this ItemTaken. The **getData**() and **updateData**() methods manage the data involved in the Agent's taking of the Item.

### ItemTaken Method Summary

| | |
|---|---|
| Evaluation | createEvaluation(Type evaluationType) |
| void | deleteEvaluation(Id evaluationType) |
| Serializable | getData() |
| String | getDisplayName() |
| EvaluationIterator | getEvaluations() |
| EvaluationIterator | getEvaluationsByType(Type evaluationType) |
| Id | getId() |
| Item | getItem() |

| | |
|---|---|
| SectionTaken | getSectionTaken() |
| void | updateData(Serializable data) |
| void | updateDisplayName(String displayName) |

## org.osid.assessment.Section

The Section is the middle-level organizing structure for the Assessment. Sections are parts of other Sections or of an Assessment and Sections contain other Sections or Items or both. As with an Assessment, the Section can order any Sections it contains. Sections can have their own data that might provide instructions or information for the application on how the Section should operate.

A Section is created through the Manager. That create method accepts a name, description, and SectionType. The create method's implementation will provide the unique Id for the Section, set the initial values for the DisplayName and Description, and set the SectionType. The **getDisplayName**(), **getDescription**(), **updateDisplayName**(), **updateDescription**(), **getId**(), and **getSectionType**() methods play the same roles as elsewhere accessing and setting these properties. Sections can be added, removed, ordered, and enumerated with **addSection**(), **removeSection**(), **orderSections**(), and **getSections**(), respectively. Items can be added, removed, ordered, and enumerated with **addItem**(), **removeItem**(), **orderItems**(), and **getItems**(), respectively. The data for this Section is accessed and set through **getData**() and **updateData**(), respectively.

### Section Method Summary

| | |
|---|---|
| void | addItem(Item item) |
| void | addSection(Section section) |
| Serializable | getData() |
| String | getDescription() |
| String | getDisplayName() |
| Id | getId() |
| ItemIterator | getItems() |
| PropertiesIterator | getProperties() |
| Properties | getPropertiesByType(Type propertiesType) |
| TypeIterator | getPropertyTypes() |
| SectionIterator | getSections() |
| Type | getSectionType() |
| void | orderItems(Item[] items) |
| void | orderSections(Section[] sections) |
| void | removeItem(Id itemId) |

| | |
|---|---|
| void | removeSection(Id sectionId) |
| void | updateData(Serializable data) |
| void | updateDescription(String description) |
| void | updateDisplayName(String displayName) |

## org.osid.assessment.SectionTaken

The SectionTaken, as with the AssessmentTaken and ItemTaken, includes the Section that was taken, any Evaluations, and the data involved. The **getSection**() method returns the Section; **createEvaluation**(), **deleteEvaluation**(), and **getEvaluations**() manage the Evaluations; and **getData**() and **updateData**() access and set the data, respectively. As with AssessmentTaken, the SectionTaken includes the fixed enumeration of Items actually taken which is returned by **getItemsTaken**(). The **getAssessmentTaken**() method returns the AssessmentTaken for this SectionTaken().

### SectionTaken Method Summary

| | |
|---|---|
| Evaluation | createEvaluation(Type evaluationType) |
| ItemTaken | createItemTaken(Item item) |
| void | deleteEvaluation(Id evaluationId) |
| void | deleteItemTaken(Id itemTakenId) |
| AssessmentTaken | getAssessmentTaken() |
| Serializable | getData() |
| String | getDisplayName() |
| EvaluationIterator | getEvaluations() |
| EvaluationIterator | getEvaluationsByType(Type evaluationType) |
| Id | getId() |
| ItemIterator | getItemsTaken() |
| Section | getSection() |
| void | updateData(Serializable data) |
| void | updateDisplayName(String displayName) |

## org.osid.assessment.<*Object type*> Iterator

Iterators return a set of elements of a specific type, one at a time. The purpose of all Iterators is to offer a way for OSID 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 included in this OSID:

- AssessmentIterator
- AssessmentPublishedIterator
- AssessmentTakenIterator
- EvaluationIterator
- ItemIterator
- ItemTakenIterator
- SectionIterator
- SectionTakenIterator

Type iterators, such as those used for Assessment, Evaluation, and Item and Section Types are defined in org.osid.shared.TypeIterator.

## org.osid.assessment.AssessmentException

The service definitions make use of Exceptions as a mechanism for responding to error or unusual conditions. Exceptional circumstances are signaled by the implementation throwing an exception, which is **AssessmentException** in the case of the Assessment OSID. The Exception contains a message that is a String. 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. The following message Strings are defined in AssessmentException:

UNKNOWN_SECTION

UNKNOWN_ITEM

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.