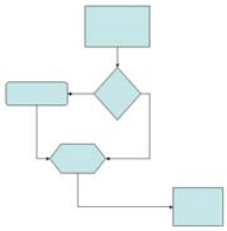
Open Service Interface Definition

Workflow



Document Release: 2.0
OSID version: 2.0

Summary

The Workflow Open Service Interface Definition (OSID) provides the means to define a Process comprised of Steps, each of which has Input Conditions and Output States. A Process exists to advance Work from an initial to a terminal Step. This advance is affected by WorkEvents that are performed as part of a Step, the result of which is a new Step Output State. WorkEvents are the result of Agents performing a specific Role in the Process.

An implementation of the Workflow OSID supports a set of possible Input Conditions, Output States and the algorithmic support determine which Steps are available to be acted upon by which Agents at any specific point in the Process.

Part of the Service is intended to define the Process and its Steps. Other parts of the Service are intended to capture the WorkEvents of a user of the Process. Separate applications for designing and using Workflow are likely to be written, each of which will rely on one or more OSID implementations.

The Workflow OSID provides an application, or set of applications with a means for coordinating and managing workflow based on some predetermined logic, among one or more actors. Abstracting and separating the workflow from the application, allows the workflow logic to be altered without much change to the application. Common tools for displaying, monitoring, and maintaining workflow could be used in conjunction with the application, saving the application from delivering this functionality.

Service Definition

An examination of an Open Service Interface Definition (OSID) usually begins with the Manager. All Managers[1] provide the way to create the objects that implement the principal interfaces in the Service. Before discussing the Manager in detail, we will review the intended role of the Workflow Service overall.

Workflow is intended to characterize activity that has a specific beginning and end and can be decomposed into discrete units. Further, a unit is not acted upon until some other unit on which it depends has reached some state.

Definitions:

- Process – a set of Steps including an Initial Step and zero or more other Steps. There is at least one Terminal Step, which can also be the initial Step. When Work reaches a Terminal Step it is Complete.

- Step – a discrete activity in a Process and associated with some Role. Each Step identifies its immediate predecessors. Steps have a defined set of possible Input Conditions and Output States. The implementation of the OSID defines the universe of possible Input Conditions and Output States. During Workflow design, an application uses methods in the Step to identify which subset of Input Conditions and Output States are appropriate for the Step.

- Expression – a name, description, and Type used to characterize the Input Conditions for a Step. Two simple examples are: "all immediate predecessors finished" and its negative.

- Input Conditions – a set of Expressions for a specific Step.

- Output State – a descriptive String used to characterize the sum of WorkEvents to date on a Step.

- WorkEvent – the union of an Agent, a point in time, and the setting of the Output State of a Step.

- Work – a grouping of WorkEvents associated with a Qualifier.

- Role – an activity with a specific name, description, and Type. Roles are analogous to Functions in Authorizations. For example, a Function might be to grade an Assessment, change a Course description, or add an Asset to a Digital Repository.

- Qualifier – the object acted upon by a particular WorkEvent. Foe example, a Qualifier might be a specific Student's grade, a specific Course, or a specific Asset in a Digital Repository. Qualifiers are analogous to Qualifiers in Authorizations.Let's consider the case of a Workflow to describe the process for grading an Assessment. The initial step in such a process might be for the Student to *submit* a completed Assessment for grading. The next step might be for a grader (Teaching Assistant) to *grade* the Assessment. The next step might be for the professor to *review* and approve the grading. The next step might be for the administrative assistant to *post* the grade to the grade book. The final step

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

might be to *notify* the Student that the Assessment is graded. These Steps, taken together, are a Process. A piece of Work (the grade) is moved along the flow of the Process by the WorkEvents of several Agents: student, teaching assistant, professor, administrative assistant. At any particular junctures, there may or may not be anything for a specific Agent with a specific role to perform. For example, before the Assessment is graded by the Teaching Assistant, the Professor, in the role of Reviewer has no available work. Once the Grade Step has reached the output state of "completed", the reviewer has something to do.

Figure 1: Single Path of Workflow

Step	Predecessors	Role	Initial?	Terminal?
Submit	none	Student	yes	no
Grade	Submit	Grader	no	no
Review	Grade	Reviewer	no	no
Post	Grade	Administrator	no	no
Notify	Review, Post	Administrator	no	yes

Each Step might be defined to have one of the following two Input Conditions: *All predecessors were successfully acted upon* and *One or more predecessors has not yet been successfully acted upon*. Each Step might be defined to have one of the following two Output States: *Done* and *Not Done*.

In the case above, each Step is part of a single path of events. A variation might be where the professor can review the grade either after the grade has done their work or after the grade is in the grade book. The final step would still depend on the professor's work, but there could be two paths through the process. A process also can be described as a network with each step being a node.

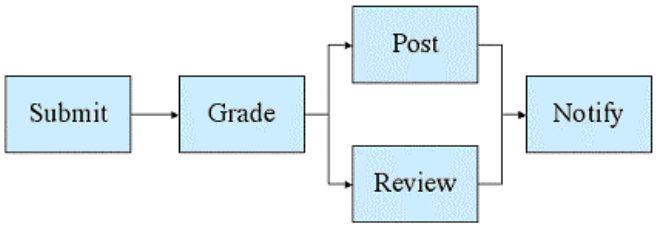


Figure 2: Network Workflow

Step	Predecessors	Role	Initial?	Terminal?
Submit	none	Student	yes	no
Grade	Submit	Grader	no	no
Review	Grade	Reviewer	no	no
Post	Grade	Administrator	no	no
Notify	Review, Post	Administrator	no	yes

With this later Workflow Process in place, let's explore what happens to a specific Work as it moves through the Process. Let's start with some Roles and Identifiers:

- Work Qualifier is Grade for Student A. Baker in English 101.
- Agent A. Baker has the Role of Student in English 101.
- Agent C. Douglas has the Role of Teaching Assistant for English 101.
- Agent E. Ferris has the Role of Professor for English 101.
- Agent G. Henderson has the Role of Administrative Assistant for English 101.

Initially, a Work is created in the Grading Process for A. Baker in English 101. Baker logs into the system and asks "What is the available work?". The Work's next Step is to Submit the Assessment. Baker does this and the Step's Output State changes from *not done* to *done*. Subsequently, C. Douglas logs in and asks for available work. Since the Output State of the Submit Step is *done* and the Input State of the Grading Step requires that this be the case before the Step is available, Douglas can Grade the Assessment. Once the Grading is done, the Process would report that there is no more available Work for either Baker or Douglas, but there is still unfinished Work for other Roles and Steps. When G. Henderson is done with the terminal Post Step, there would be no more available or unfinished Work. The Workflow Manager would report that the Work was completed.

The following are some comments on Input Conditions and Output States. In an implementation of a workflow service, there are defined input conditions (expressions) and output states. The WorkflowManager returns the set of all possible values for input conditions and output states. Each step in the workflow has a set of possible input conditions and output states drawn from the set of all possible values. These values are part of the support in a given workflow service implementation and in are part of the design of any particular process.

When working with an active process, the Work has a method to update the output state of any particular step. The Work also has a history, which is a sequence of WorkEvents. Each WorkEvent references a specific step and output state.

Suppose we have a simple process that has steps for taking a test, grading the test, and then either reviewing remedial material or being done.

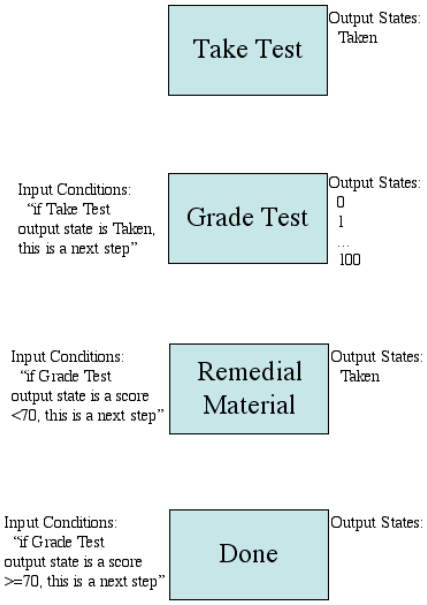


Figure 3: Sample Process

The input conditions refer to the output state of some other step. Some steps have only a single output state, if any, while the Grade Test step has a range of possible output states. This process definition can be made a little more complex by adding an input condition to the Take Test step. Input conditions are not evaluated for the initial step of a process. Thereafter, any input conditions are evaluated for the initial step as well as all other steps each time the process needs to determine next steps. An input condition for the Take Test step referencing the output state of the Remedial Material allows the process to loop back to the start.

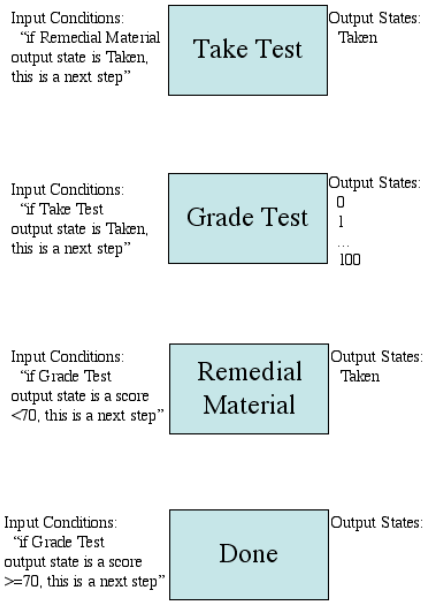


Figure 4: Illustration of Looping Back to Start of Process

When it comes time to interact with the workflow process, a history of work events captures the output state of each step. At that time, there is no longer a range of possible grades for the Grade Test output state; rather there is a single, specific grade.

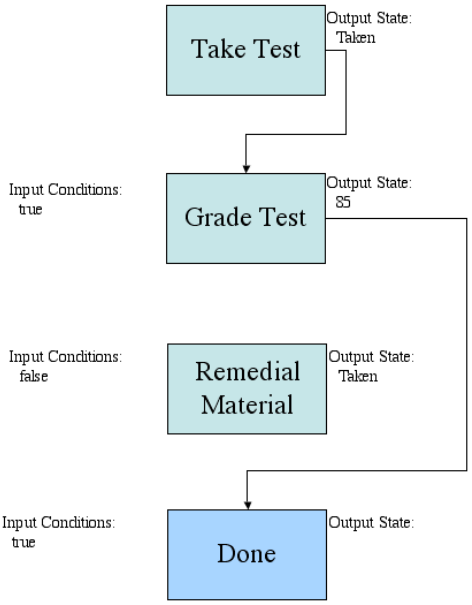


Figure 5: Flow Through Process With Specific Conditions and States

org.osid.workflow.WorkflowManager

An installation may have several implementations of WorkflowManager available. Like Managers in others O.K.I. services, WorkflowManagers with desired features can be obtained through OsidLoader. Once a WorkflowManager is available, it can create, delete, and get Processes using **createProcess**(), **deleteProcess**(), and **getProcess**() and **getProcesses**(), respectively. The **copyProcess**() method makes a new Process from a copy of all the elements in an existing Process; only the display name and description are overridden. Once the Process is copied, changes to the original do not affect the copy.

In addition to managing the Process lifecycle, the Manager returns information about InputConditions (Expressions) and OutputStates. As we have seen in other Managers that deal with objects with Types that are supported by an implementation, we have a trio: **getExpressions**() with no arguments which returns all the obejcts, **getExpressionsByType**() with a Type argument which returns only the objects of a specific Type, and **getExpressionTypes**() which returns all the supported Types.

WorkflowManager Method Summary

Process	copyProcess(Id originalProcessId, String newDisplayName, String newDescription)
Process	createProcess(String displayName, String description)
void	deleteProcess(Id processId)
WorkIterator	getCompletedWork(Id processed)
ExpressionIterator	getExpressions()

ExpressionIterator	getExpressionsByType(Type expressionType)
TypeIterator	getExpressionTypes()
StringIterator	getOutputStates()
Process	getProcess()
ProcessIterator	getProcesses()
WorkIterator	getUnfinishedWork(Id processId)
boolean	supportsDesign()
boolean	supportsMaintenance()

The following methods might be relevant for different applications.

Design a Workflow	Use a Workflow	Report on Workflows
copyProcess	getProcess	getCompletedWork
createProcess	getProcesses	getUnfinishedWork
deleteProcess		
getExpressions		
getExpressionTypes		
getOutputStates		

org.osid.workflow.Process

The Process is responsible for defining the order and dependencies among Steps and the kinds of Work. These setup methods are **createInitialStep**() which must be called once and only once, **createStep**(), **createWork**(), **deleteStep**(), **getStep**() and **getSteps**(), **getWork**(), and **getAll-Work**().

A Process, once created, can be enabled or disabled using **updateEnabled**() and tested for this state with **isEnabled**().

The Process also has methods for determining the state of Work Completed, Available, and Un-finished. Work can also be frozen using **haltWork**() and restarted using **resumeWork**().

Process Method Summary

Step	createIntialStep(String displayName, String description, Id roleId)
Step	createStep(Id predecessorStepId, String displayName, String description, Id roleId)
Work	createWork (String displayName, String description, Id qualifierId)
void	deleteStep(Id stepId)
void	deleteWork(Id workId)
WorkIterator	getAvailableWork()
WorkIterator	getAvailableWorkForRole(Id roleId)
WorkIterator	getAvailableWorkForStep(Id stepId)
String	getDescription()

String	getDisplayName()
Id	getId()
Step	getStep(stepId)
StepIterator	getSteps()
Type	getType()
WorkIterator	getUnfinishedWork()
WorkIterator	getUnfinishedWorkForRole(Id roleId)
WorkIterator	getUnfinishedWorkForStep(Id stepId)
Work	getWork(Id workId)
WorkIterator	getWorkAllWork()
void	haltWork(Id workId)
boolean	isEnabled()
void	resumeWork(Id workId)
void	updateDescription(String description)
void	updateDisplayName(String displayName)
void	updateEnabled(boolean enabled)

The following methods might be relevant for different applications.

Design a Workflow	Use a Workflow	Report / Admin Workflows
createIntialStep	createWork	getUnfinishedWork
createStep	deleteWork	haltWork
deleteStep	getAvailableWork	resumeWork
updateDescription	getDescription	updateEnabled
updateDisplayName	getDisplayName	
	getId	
	getStep	
	getSteps	
	getWork	
	getWorkAllWork	
	isEnabled	

org.osid.workflow.Step

Step is a discrete unit of activity in the Process. The circumstances of the Step can be returned through **getInputConditions**() and **getOutputStates**(). A step can add or remove an additional immediate predecessor using **addPredecessor**() and **removePredecessor**(), respectively. A single predecessor is assigned when a Process creates the Step. Steps are mapped to Func-

tions, perhaps in an Authorization system. The relevant methods are: **getRoleId**() and **updateRoleId**().

Implementations support a set of InputConditions and Output States. In the design of a Step, the subset of Input Conditions and Output States that should be used can be defined using **updateInputConditions**() and **updateOutputStates**().

There is one and only one initial step. The input conditions for the initial step are ignored when the process begins, but are evaluated each subsequent time next steps are calculated. This means the initial step will be the next step at least once. There are one or more terminal steps.

Step Method Summary

void	addPredecessor(Id stepId)
StepIterator	getSuccessors()
String	getDescription()
String	getDisplayName()
Id	getId()
ExpressionIterator	getInputConditions()
StringIterator	getOutputStates()
StepIterator	getPredecessors()
Id	getRoleId()
boolean	isInitial()
boolean	isTerminal()
void	removePredecessor(Id stepId)
void	updateDescription(String description)
void	updateDisplayName(String displayName)
void	updateInputConditions(Expression[] inputConditions)
void	updateOutputStates(String[] outputStates)
void	updateRoleId(Id roleId)

The following methods might be relevant for different applications.

Design a Workflow	Use a Workflow	Report / Admin Workflows
addPredecessor	getDescription	
getId	getDisplayName	
updateDescription	getInputConditions	
updateDisplayName	getOutputStates	
updateInputConditions	getPredecessors	
updateOutputStates	getRoleId	
updateRoleId	getSuccessors	

org.osid.workflow.Expression

An Expression acts as an input condition for a Step.

Expression Method Summary

String	getDescription()
String	getDisplayName()
Type	getType()

org.osid.workflow.Work

Work holds a group of WorkEvents that are returned by **getHistory**(). The Work is associated with a Qualifier, perhaps in an Authorization system. In this manner, it is Work in regard to some specific thing. For each thing in a system, say each Assessment to be graded, there is a separate body of Work.

The implementation will support a set of these Output States. These are returned by the **getOutputStates**() method in the Manager. The Work **updateStepOutputState**() method updates the Output State of this Step to one of those defined for the Step.

Work Method Summary

String	getDescription()
String	getDisplayName()
WorkEventIterator	getHistory()
Id	getId()
void	updateDescription(String description)
void	updateDisplayName(String displayName)
StepIterator	getNextSteps()
StepIterator	getNextStepsForRole(Id roleId)
PropertiesIterator	getProperties()
Properties	getPropertiesByType(Type propertiesType)
TypeIterator	getPropertyTypes()
Id	getQualifierId()
void	updateStepOutputState(Id stepId, String outputState)

org.osid.workflow.WorkEvent

A WorkEvent is the union of an Agent, the Step the Agent acts upon, the time and date of the WorkEvent, and the resultant OutputState for the affected Step. This is “get-only” information. Capturing the WorkEvent and setting these values is the responsibility of the implementation and outside the scope of the OSID.

WorkEvent Method Summary

Id	getAgentId()
String	getOutputState()
Step	getStep()
long	getTimeStamp()

org.osid.workflow.*Object* Iterators

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:

- ProcessIterator
- StepIterator
- ExpressionIterator
- WorkIterator
- WorkEventIterator

Type iterators, such as that used for Expression Types are defined in osid.shared.TypeIterator.

String iterators, such as that used for Output States are defined in osid.shared.StringIterator.

org.osid.workflow.WorkflowException

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

Exception Message Summary

Constant	Message String
CIRCULAR_OPERATION	Circular operation
ALREADY_ADDED	Already added
UNKNOWN_EXPRESSION	The Expression is not known to the system
UNKNOWN_OUTPUT_STATE	The Output State is not known to the system

INVALID_NETWORK	Invalid network
NOT_HALTED	Not halted

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 sub-class 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.