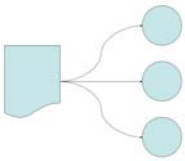
Open Service Interface Definition

User Messaging



Document Release: 2.0
OSID version: 2.0

Summary

The Usermessaging Open Service Interface Definition (OSID) provides a means of sending Messages, subscribing to receive Messages, and receiving Messages. Each Message has a Type and Topic[1] and what Messages are Received can be filtered by Type and Topic. Topic is not necessarily the subject of the Message. Messages can be sent to a specific subset of Subscribers or to all.

Usermessaging is a general Service intended to address underlying mail, instant messaging or chant, and threaded discussion systems.

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 discussing the Manager in detail, we will review the intended role of the Usermessaging Service overall.

The intent of the Usermessaging OSID is that it can support at least the common applications of mail, instant messaging, and threaded discussion boards or databases. Let's examine each of these three applications in turn.

Mail – With an underlying mail system, Messages likely will be encoded with header and body, perhaps using MIME. The Message Content can be set to the entire mail message with an appropriate MessageType. Mail systems will have a variety of ways to send mail and know what happened to it. Delivery confirmation or failure information can be returned to the send as another Message. The OSID has a sin-

[1] The subject can be embedded in the Content of the Message. The intent is that the Topic is the same Topic used in CourseManagement to refer to an area of interest. This allows people to subscribe to areas of interest.

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

gle class of intended recipient rather than, for example, a set of primary addressees, people who are copied, and people who are blind copied. This could be handled either by sending information separately or more simply by embedding that information into the header portion of the Message Content. This can be managed through Message composing and reading applications. Mail may be sent through a gateway and received using a Post Office or other models. These can be accommodated by the send and receive methods and Delivery Types. Mail systems preserve Messages, often on a server, a client, or both. Deleting a Message might be interpreted as deleting it from one's own mailbox on the Server.

Instant Messaging or Chat – With an underlying instant messaging or similar system, Messages may be plain text. The Message Content can be set to this text with an appropriate Message Type. Instant messages may be sent once and only received by those currently on-line and accepting messages. Agents could subscribe and unsubscribe as they came on-line or off-line or that information could be stored on a server as a profile. The subscribers list returned could only include Agents that are on-line. Instant messaging systems may not store messages. Deleting a Message may not be necessary as the Message text may simply be appended to a chat script and stored separately.

Discussions – With an underlying discussion board or discussion database system, Messages may be sent to the discussion and stored centrally. Perhaps each Topic is a discussion thread. Thread might be organized using the Hierarchy OSID. The Message Content might be anything. Sending a Message becomes posting a Message to a discussion thread. Sending to a new Topic might create a new thread. There is no notion of message delivery here. Receiving a Message may become viewing the discussion thread or new postings to it. Deleting a Message may not be relevant but the administrator may want to purge inappropriate or out of date messages.

org.osid.usermessaging.UsermessagingManager

An installation may have several implementations of UserMessagingManager available. Like managers in others O.K.I. services, UserMessagingManagers with desired features can be obtained through OsidLoader. Once a UserMessagingManager is available, it can be used to manage subscribers, manage Messages, and send and receive Messages. The current user[3] can subscribe by Topic to the Service using the **subscribe**() method. If authorized, one can subscribe any Agent (or Group) for a Topic as well. The **unsubscribe**() and **unsubscribeAll**() removes the current user as subscriber to a particular Topic or all Topics, respectively.

Messages can be sent to an individual Agent (or Group which is a subclass of Agent), a set of Agents, or all subscribers. The **send**()and **sendToAll**() methods support this. Note the intent is that the sender does not receive an Exception if the Message is not received for any reason. Any delivery-related information could be provided to the sender by the implementation in the form of a Message from the implementation (perhaps as an administration agent) to the original sender. Delivery Types can support this. Messages are received for any Topic or only for specific Topics or MessageTypes. The **receive**(), **receieveForMessageType**(), and **receiveForTopic**() methods support this.

The Message and Delivery Types an implementation supports are returned by **getMessageTypes**() and **getDeliveryTypes**(), respectively.

UsermessagingManager Method Summary

void	deleteMessage(Message message)
void	deleteMessages(long before, Type messageType, Sting topic)
TypeIterator	getDeliveryTypes()
TypeIterator	getMessageTypes()
IdIterator	getSubscribers()

[3] The Authentication OSID provides a method to map the current authenticated user to an Agent Id.

IdIterator	getSubscribersByTopic(String topic)
StringIterator	getTopics()
void	purgeMessage(Message message)
MessageIterator	receive()
MessageIterator	receiveForMessageType(Type messageType)
MessageIterator	receiveForTopic(String topic)
void	send (Id[] agents, Serializable content, Type messageType, Type deliveryType, String topic)
void	sendToAll(Serializable content, Type messageType, Type deliveryType, String topic)
void	subscribe(Id agentId, String topic)
void	unsubscribe(Id agentId, String topic)
void	unsubscribeAll(Id agentId)

org.osid.usermessaging.Message

A Message contains “get-only” attributes. The Message is created through the Manager when a **send**() method is called. All the attributes are set at that time by the implementation. The Content of the Message can be simple text or a complex encoding such as MIME.

Message Method Summary

Serializable	getContent()
Type	getDeliveryType()
long	getMessageTimestamp()
Type	getMessageType()
Id	getSender()
String	getTopic()

org.osid.usermessaging.*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 SID 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:

- MessageIterator

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

String iterators, such as that used for Topics are defined in org.osid.shared.StringIterator.

org.osid.usermessaging.UsermessagingException

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

Exception Message Summary

Constant	**Message String**
NOT_SUBSCRIBED	Not currently subscribed to this Topic
UNKNOWN_TOPIC	The Topic is not know to the system

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.