Author:
Last Modified: 2004-10-02T19:45:00Z
By:

Table of Contents

Overview

The methods in this document define the functionality for storing, deleting, updating and retrieving experiment records. This API, like the *Service Broker Administrative API*, is intended to specify, at least initially, an internal but self-contained library accessed as part of the Service Broker process. Eventually, it may be useful to expose some or all of the methods described below as web service calls so that the Lab Server or Client may access the Experiment Storage Service directly.

The design recognizes that much of an experiment record is domain-dependent and, therefore, meaningless (opaque) to the Service Broker though intelligible to the Lab Server and the user's Lab Client. More specifically, the record for a completed experiment consists of three such strings opaque to the Service Broker:

- **`LabConfiguration,`**
- **`ExperimentSpecification, and`**
- **`ExperimentResult.`**

Users will need to be able to search for experiment records and administer them, e.g. delete or archive them, independent of the Lab Server or Client. This implies that the Service Broker must understand certain semantics common to all experiment types. We have segregated the fields that support these semantics in the transparent (to the Service Broker) type, **`ExperimentInformation`** (see below). Conversely, we have decided that the Experiment Storage API should have no responsibility for enforcing administrative policy. So, this API treats the fields of the **`ExperimentInformation`** object, with the exception of **`experimentID`**, as if they were pure data fields. For instance, this API does not check the referential integrity of the **`userID`** field in **`ExperimentInformation`**. The **`User`** class is an administrative concept. When a user is deleted, the decision whether to purge or archive a user's experiments is an administrative decision. This API will allow the Service Broker or another service to enforce any chosen policy.

The Service Broker architecture needs to be able to store and retrieve three other classes of information:

- convenience or preference items associated with a particular user and software client for accessing a particular lab server;

- unfinished experiment specifications; and
- sets of arbitrary binary data (BLOBs) associated with experiment records, e.g. images and video streams.

The first two classes are not associated with an **`experimentID`** or with any interaction with the Lab Server. Because of this, we have chosen to support the first two categories of functionality via the **`ClientData`** methods of the *Service Broker Administrative API*. The third category, however, does describe functionality that extends the basic storage capabilities for experiments and is linked to the concept of **`experimentID`**. This functionality was not implemented in the first version of this API. Extensive support for BLOB storage is provided by the second generation general Experiment Storage Service (created to support interactive as well as batched experiments). The developer should consult the Experiment Storage Service Specification (2004-08-18 or later).

As in the *Service Broker Administrative API*, none of the methods described below will check whether the current user has permission to call the method. We assume that such checks are being performed by the trusted code that invokes these methods using mechanisms detailed in the *Service Broker Authorization API*.

The ExperimentInformation Class

The transparent **`ExperimentInformation`** object associated with each experiment record contains all the information about the experiment that is accessible to the Service Broker. Most of the fields of the **`ExperimentInformation`** class are fixed, for example **`experimentID`**, **`effectiveGroupID`**, **`userID`**, etc. They are permanently set by the time the lab server executes the experiment. The Service Broker can, however, change the **`annotation`** field for the client after experiment execution. This allows the user to supply a **`string`** title or caption to a particular experiment.

Two further fields, **`xmlResultExtension`** and **`xmlBlobExtension`**, provide an XML-based extension mechanism. These transparent XML strings define sets of attribute-value pairs that are set by the Lab Server and read by the Service Broker. These fields should conform to XML Schemas defined by the lab server for a particular type of experiment. Our intent is to allow searches for experiments and blobs

by attribute, but this capability will not be implemented in the first version of the Data Storage API.

```
public struct ExperimentInformation
{
     public readonly int experimentID;
     public readonly int labServerID;
     public readonly int userID;
     public readonly int effectiveGroupID;
     public readonly string/DateTime submissionTime;
     public readonly string/DateTime completionTime;
     public readonly string/DateTime expirationTime;
     public readonly double minTimeToLive;
     public readonly int priorityHint;
     public readonly int statusCode;

     public readonly string [] validationWarningMessages;
     public readonly string    validationErrorMessage;
     public readonly string [] executionWarningMessages;
     public readonly string    executionErrorMessage;

     public          string annotation;
     public readonly string xmlResultExtension;
     public readonly string xmlBlobExtension;
}
```

While the following Experiment class is never formally defined, conceptually it represents the complete record of an experiment.

```
public struct Experiment
{
     ExperimentInformation  information;
     public readonly string labConfiguration;
     public readonly string experimentSpecification;
     public readonly string experimentResult;
}
```

Service Broker Data Storage API Architecture

Methods

Configuration Methods

SetResultXMLExtensionSchema

Purpose:

/* Identify the XML schema used to validate the **xmlResultExtension** data member of the **ExperimentInformation** class on the Service Broker for experiments executed on the lab server identified by **labServerID**. Any change to this field should be upwardly compatible. That is, if the **xmlResultExtension** schema is changed, the new schema should successfully validate all XML documents that were successfully validated by its predecessor. */

Arguments:

int labServerID
/* Identifies the lab server to which the XML result extension applies. */

string URL
/* The URL of the XML schema document defining the new validating schema. */

Returns:

void none

SetBlobXMLExtensionSchema

Purpose:

/* Identify the XML schema used to validate the **xmlBlobExtension** data member of the **ExperimentInformation** class on the Service Broker for experiments executed on the lab server identified by **labServerID**. Any change to this field should be upwardly compatible. That is, if the **xmlBlobExtension** schema is changed, the new schema should successfully validate all XML documents that were successfully validated by its predecessor. */

Arguments:

`int labServerID`
/* Identifies the lab server to which the XML BLOB extension applies. */

`string URL`
/* The URL of the XML schema document defining the new validating schema. */

Returns:

`void none`

Experiment Record Methods

CreateExperiment

Purpose:

/* Creates an empty experiment record identified by a unique experimentID, which is returned. */

Arguments:

`int userID`
/* Identifies the user, who becomes the owner of all records pertaining to this experiment. */

`int effectiveGroupID`
/* Identifies the effective group, which sets the permissions under which the experiment will execute. */

Returns:

`int experimentID`
/* The unique experimentID. */

SaveExperimentSpecification

Purpose:

/* Records an experiment specification. */

Arguments:

`int experimentID`
/* Identifies the experiment. */

`int labServerID`

/* Identifies the lab server on which the experiment will execute. */

int userID
/* Identifies the user. */

int effectiveGroupID
/* Identifies the effective group, which sets the permissions under which the experiment will execute. */

string experimentSpecification
/* An opaque string generated by the Lab Client to specify an experiment. */

string annotation
/* An optional user-defined annotation; null if not used. */

Returns:

void none

RetrieveExperimentSpecification

Purpose:

/* Retrieves a previously saved experiment specification. */

Arguments:

int experimentID
/* Identifies the experiment. */

Returns:

string experimentSpecification
/* The **experimentSpecification**, a domain-dependent experiment specification originally created by the Lab Client. */

SaveSubmissionReport

Purpose:

/* Saves the portions of the **SubmissionReport** returned from the Lab Server that belong in the ExperimentInformation object. */

Arguments:

int experimentID

/* Identifies the experiment. */

SubmissionReport sReport
/* The **SubmissionReport** object returned by the Lab Server in response to a Submit call. See the *Service Broker to Lab Server API*. */

Returns:

void none

SaveResultReport

Purpose:

/* Saves the experiment result returned from Lab Server. */

Arguments:

int experimentID
/* Identifies the experiment on the Service Broker. */

ResultReport rReport
/* The **ResultReport** object returned by the Lab Server in response to a **RetrieveResult** call. See the *Service Broker to Lab Server API*. */

Returns:

void none

RetrieveExperimentResult

Purpose:

/* Retrieves the result from a previously executed experiment. */

Arguments:

int experimentID
/* Identifies the experiment. */

Returns:

string experimentResult
/* The **experimentResult**, an opaque string generated by the Lab Server to describe the result of a previously executed experiment. */

SaveLabConfiguration

Purpose:

/* Records the lab configuration for a particular experiment. */

Arguments:

`int experimentID`
/* Identifies the experiment. */

`string labConfiguration`
/* The lab configuration to be saved. **`labConfiguration`** is an opaque string included in the **`ResultReport`** by the Lab Server to specify the configuration in which an experiment is executed. */

Returns:

`void none`

RetrieveLabConfiguration

Purpose:

/* Retrieves a previously saved lab configuration for a particular experiment. */

Arguments:

`int experimentID`
/* Identifies the experiment. */

Returns:

`string labConfiguration`
/* The **`labConfiguration`**, an opaque string included in the **`ResultReport`** by the Lab Server to specify the configuration in which an experiment is executed. */

SaveExperimentAnnotation

Purpose:

/* Saves or modifies an optional user defined annotation to the experiment record. */

Arguments:

int experimentID
/* Identifies the experiment. */

string annotation
/* An optional user-defined annotation to the experiment. */

Returns:

string previousAnnotation
/* The previous annotation or null if there wasn't one. */

RetrieveExperimentAnnotation

Purpose:

/* To retrieve a previously saved experiment annotation. */

Arguments:

int experimentID
/* Identifies the experiment. */

Returns:

string experimentAnnotation
/* The annotation, a string originally created by the user via the Lab Client. */

RemoveExperiments

Purpose:

/* Deletes all record of the experiments specified by an array of experiment IDs. */

Arguments:

int[] experimentIDs
/* An array that identifies the experiments to be deleted. */

Returns:

int[] unRemovedExperiments
/* An array containing the subset of the specified experiment IDs for which the delete operation failed. */

GetExperimentInformation

Purpose:

/* To retrieve experiment metadata for experiments specified by an array of experiment IDs. */

Arguments:

int[] experimentIDs
/* An array that identifies the experiments whose metadata is to be retrieved. */

Returns:

ExperimentInformation[] experiments
/* An array of instances of the specified **ExperimentInformation** objects. */

ModifyExperimentOwner

Purpose:

/* Allows an authorized user to modify ownership of an experiment for administrative purposes. */

Arguments:

int experimentID
/* Identifies the experiment whose ownership will be modified. */

int newUserID
/* Designates the new owner of the experiment. */

Returns:

void none

ModifyExperimentStatus

Purpose:

/* Updates the status code of an experiment. */

Arguments:

int experimentID
/* Identifies the experiment whose status will be modified. */

int statusCode
/* Designates the new status code of the experiment. */

Returns:

`bool result`
/* Returns **`true`** if the experiment was successfully updated; **`false`** otherwise. */

Search Methods

```
public struct Criterion
      {
            public readonly string attribute
            public readonly string predicate
            public readonly string value
      }
```

The **attribute** field must be one of a set of predefined strings specifying an attribute of an experiment. The **predicate** field must also be a predefined string that specifies a predicate search condition, e.g. "**equals**", "**before**". The value field can be any string. The **Criterion** specified by the triple { "**userName**", "**equals**", "**jud**"} would specify a search condition that would only look for experiments specified by the user with the ID "**jud**".

FindExperimentIDs

Purpose:

/* Retrieves the **experimentID**s for all experiments that match the conditions expressed by the **Criterion** argument array. The conditions expressed by each criteria element are ANDed together to determine the final experiment selection. */

Arguments:

Criterion[] criteria
/* An array of **Criterion** objects that specifies a search condition. */

Returns:

int[] experimentIDs
/* The array of all integer **experimentID**s whose associated experiments matched the conjunction of the conditions expressed by the criteria argument. */