The date structure:

```
public class TimePeriod
         {
            DateTime startTimeField;

            DateTime endTimeField;
         }
```

User-side Scheduling Server :

```
public struct UssCredentialSet
         {
            int credentialSetID;
            String serviceBrokerID;
            String serviceBrokerName;
            String groupName;
         }


public struct Reservation
         {
            int reservationID;
            String userName;
            int credentialSetID;
            DateTime startTime;
            DateTime endTime;
            int experimentInfoID;


         }

public struct USSPolicy
```

```
        {
           int ussPolicyID;
           int experimentInfoID;
           String rule;
           int credentialSetID;
        }

public struct UssExperimentInfo
        {
           int experimentInfoID;
           String labServerID;
           String labServerName;
           String labClientVersion;
           String labClielntName;
           String providerName;
           String lssID;
        }

public struct LSSInfo
        {
          int lssInfoID;
          String lssID;
          String lssName;
          String lssURL;
        }
```

Lab-side Scheduling Server:

```
    public struct LssCredentialSet
        {
           int credentialSetID;
           String serviceBrokerID;
           String serviceBrokerName;
           String groupName;
           String ussID;
        }

    public struct ReservationInfo
        {
           int reservationInfoID;
           int credentialSetID;
           DateTime startTime;
           DateTime endTime;
           int experimentInfoID;
        }

    public struct TimeBlock
        {
```

```
            int timeBlockID;
            int credentialSetID;
            DateTime startTime;
            DateTime endTime;
            String labServerID;
            int recurrenceID
        }

     public struct LssExperimentInfo
        {
            int experimentInfoID;
            String labServerID;
            String labServerName;
            String labClientVersion;
            String labClielntName;
            String providerName;
            int quantum;
            int prepareTime;
            int recoverTime;
            int minimumTime;
            int earlyArriveTime
        }


    public struct LSSPolicy
        {
            int lssPolicyID;
            int credentialSetID;
            string rule;
            int experimentInfoID;
}

     public struct PermittedExperiment
        {
            int permittedExperimentID;
            int experimentInfoID;
            int recurrenceID;
        }

     public struct USSInfo
        {
            int ussInfoID;
            String ussID;
            String ussName;
            String ussURL;
        }
```

```
public struct Recurrence
  {
      int recurrenceID;
      int credentialSetID;
      string recurrenceType;
      DateTime recurrenceStartDate;
      DateTime recurrenceEndDate;
      TimeSpan recurrenceStartTime;
      TimeSpan  recurrenceEndTime;
      String labServerID;
   }
```

Internal API

User-side Scheduling Server

User Side Scheduling Policy Management Methods

AddUSSPolicy

Purpose:

/*Add user side scheduling policy that governs whether a reservation request to execute an experiment at a certain time will be accepted from a student with a particular credential set. */

Arguments:

`string groupName`
/* the name of the group whose members need to obey this policy*/

`string serviceBrokerID`
/* the GUID identifying the service broker whose domain the group above belongs to*/

`int experimentInfoID`
/* the unique ID identifying the experiment that the policy applies to*/

`string rule`
/* the rule set that governs whether a reservation request to execute an experiment at a certain time will be accepted from a student with a particular credential set. */

Returns:

`int ussPolicyID`

/*The unique ID which identifies the user side scheduling policy added. >0 was successfully added; ==-1 otherwise */

RemoveUSSPolicies

Purpose:

/*Delete the user side scheduling policies specified by the ussPolicyIDs. */

Arguments:

`int[] ussPolicyIDs`
/* An array of user side scheduling policyIDs specifying the policies to be removed*/

Returns:

`int[] unremovedUssPolicyIDs`
/*An array of ints containing the IDs of all UssPolicies not successfully removed, i.e., those for which the operation failed. */

ModifyUSSPolicy

Purpose:

/*Updates the data fields for the user side scheduling policy specified by the ussPolicyID; note ussPolicyID may not be changed; */

Arguments:

`int ussPolicyID`
/* the ID identifying the policy whose data is being changed */

`int experimentInfoID`
/* the new unique ID identifying the experiment that the policy applies to*/

`string rule`
/* the new rule set that governs whether a reservation request to execute an experiment at a certain time will be accepted from a student with a particular credential set. */

`int credentialSetID`
/* the ID identifying the credential set whose data is being changed */

Returns:

`Bool modified`
/* true if modified successfully, otherwise false*/

ListUSSPolicyIDsByGroup

Purpose:

/*Enumerates all IDs of the User side Scheduling Policies applying for the users with a particular credential set identified by the combination of groupName and serviceBrokerID */

Arguments:

`string groupName`
/* the name identifying the group whose policies are to be listed */

`string serviceBrokerID`
/* the GUID identifying service broker whose domain the group above belongs to */

Returns:

`int[] ussPolicyIDs`
/* An array of ints containing the IDs of all the User-side Scheduling Policies applying for the users with a particular credential set * /

GetUSSPolicies

Purpose:

/*Returns an array of the immutable UssPolicy objects that correspond to the supplied ussPolicy IDs. */

Arguments:

`int[] ussPolicyIDs`
/* The IDs identifying the ussPolicies whose information is being requested. */

Returns:

`USSPolicy[] policies`
/* An array of immutable objects describing the specified user side scheduling policies; if the nth ussPolicyID does not correspond to a valid policy, the nth entry in the return array will be null.*/

Reservation Management Methods

AddReservation

Purpose:

/*Add reservation by user. */

Arguments:

`string userName`
/* the name of the user who is making the reservation*/

`string serviceBrokerID`
/* the GUID identifying the service broker whose domain the user above belongs to*/

`string groupName`
/* the name of the group that the user above belongs to*/

`int experimentInfoID`
/* the unique ID identifying the experiment that the user want to reserve*/

`DateTime startTime`
/*the startTime of the reservation. note startTime is time in UTC */

`DateTime endTime`
/* the endTime of the reservation. note endTime is time in UTC */

Returns:

`int reservationID`
/*The unique ID which identifies the reservation added by the user. >0 was successfully added, in order for this happen, the reservation being added need be validated by calling **ValidateReservation()**; ==-1 otherwise. */

RemoveReservations

Purpose:

/*Delete the reservations specified by the reservation IDs. */

Arguments:

`int[] reservationIDs`
/* An array of reservarion IDs specifying the reservations to be removed*/

Returns:

`int[] unremovedReservationIDs`
/*An array of ints containing the IDs of all reservations not successfully removed, i.e., those for which the operation failed. */

ModifyReservation

Purpose:

/*Updates the data fields for the reservation specified by the reservationID; note reservationID may not be

changed; and since the ServiceBrokerID, the UserName and the GroupName come from user's credential set, these fields may also not be changed*/

Arguments:

int reservationID
/* the ID identifying the reservation whose data is being changed */

int experimentInfoID
/* the unique ID identifying the new experiment that the user want to reserve*/

DateTime startTime
/* the new startTime of the reservation. note startTime is time in UTC */

DateTime endTime
/* the new endTime of the reservation. . note endTime is time in UTC */

Returns:

bool modified.
/*true if reservation was successfully modified, in order for this happen, the reservation being modified need be validated by calling ValidateReservation(); ==false otherwise.*/

ListReservationIDsByUser

Purpose:

/*Enumerates all IDs of the reservations on a particular experiment made by a particular user identified by the combination of userName and serviceBrokerID */

Arguments:

string userName
/* the name of the user whose reservations are to be listed */

string serviceBrokerID
/* the GUID identifying service broker that the user above belongs to */

int experimentID
/* the unique ID identifying the experiment on which the user makes reservation */

Returns:

int[] reservationIDs

/* An array of ints containing the IDs of all the reservations made on a particular experiment by the specified user */

ListReservationIDsByGroup

Purpose:

/*Enumerates all IDs of the reservations with a credential sets identified by the combination of groupName and serviceBrokerID */

Arguments:

`string groupName`
/* the name of the group whose members' reservations are to be listed*/

`string serviceBrokerID`
/* the GUID identifying service broker that the group above belongs to */

Returns:

`int[] reservationIDs`
/* An array of ints containing the IDs of the reservations with a credential sets identified by the combination of groupName and serviceBrokerID */

ListReservationIDsByLabServer

Purpose:

/*Enumerates all IDs of the reservations happen on the particular service broker during a particular time period, which include the reservation part of which will happen during the time period */

Arguments:

`string labServerID`
/* the GUID of the lab server ID on which the reservations are made*/

`DateTime startTime`
/* the start time(UTC) of the time period, the IDs of reservations in which are queried*/

`DateTime endTime`
/* the end time(UTC) of the time period, the IDs of reservations in which are queried */

Returns:

int[] reservationIDs
/* An array of ints containing the IDs of the reservations made on the particular lab server during a particular time period */

GetReservations

Purpose:

/*Returns an array of the immutable reservation objects that correspond to the supplied reservation IDs. */

Arguments:

int[] reservationIDs
/* The IDs identifying the reservations whose information is being requested. */

Returns:

Reservation[] reservations
/* An array of immutable objects describing the specified reservations; if the nth reservationID does not correspond to a valid reservation, the nth entry in the return array will be null.*/

SelectReservation

Purpose:

/* to select reservation accorrding to given criterion */

Arguments:

String userName
/* the user who made the reservation */

int experimentInfoID
/* The IDs identifying experiment of the reservation. */

int credentialSetID
/* The IDs identifying the credential set the user belongs to */

DateTime timeAfter
/* the UTC time that the selected reservation starts later than. */

DateTime timeBefore
/* the UTC time that the selected reservation starts earlier than. */

Returns:

Reservation[] reservations
/* An array of immutable objects describing the specified reservations */

RevokeReservation

Purpose:

`/* remove all the reservation for certain lab server being covered by the revocation time, which include the reservation part of which will happen during the revocation time */`

Arguments:

String labServerID
/* the ID of the labserver whose reservation is to be revoked*/

DateTime startTime
/* the start time of the revocation time, the local time of the USS*/

DateTime endTime
/* the end time of the revocation time, the local time of the USS*/

Returns:

Boolean revoked
/* true if all the reservations have been removed successfully*/

RedeemReservation

Purpose:

/* Returns the reservation that can be used now by a particular user to execute a particular experiment*/

Arguments:

String userName
/* the name of the user who made the reservation*/

String serviceBrokerID
/* the ID of the service broker which manage the user's information*/

String labClientName
/* the lab client of the experiment for which the reservation is made for*/

String labClientVersion

/* the lab client version of the experiment for which the reservation is made for*/

Returns:

Reservation reservation
/* the reservation that can be used now by a particular user to execute a particular experiment, if the reservation == null, there is no reservation can be used now by a particular user to execute a particular experiment*/

RedeemReservation

Purpose:

/* Return the time span the user should wait till the start time of the reservation */

Arguments:

int reservationID
/* the reservation ID to be redeemed*/

Returns:

TimeSpan timespan
/*the time span the user should wait till the start time of the reservation */

Experiment Information Management Methods

AddExperimentInfo

Purpose:

/* Add information of a particular experiment*/

Arguments:

string labServerID
/* the GUID identifying the lab server in which the experiment is executed */

string labServerName
/* the name of the lab server in which the experiment is executed */

string labClientVersion
/* the version of the lab client of the experiment*/

string labClientName

/* the name of the lab client of the experiment */

string providerName
/* the name of the provider of the lab server */

string lssID
/* the GUID identifying the lab side scheduling server which manages the reserve information of this experiment */

Returns:

int experimentInfoID
/*The unique ID which identifies the experiment added. >0 was successfully added, ==-1 otherwise. */

ModifyExperimentInfo

Purpose:

/*Updates the data fields for the ExperimentInfo specified by the ExperimentInfoID; note ExperimentInfoID may not be changed*/

Arguments:

int experimentInfoID
/* the unique ID identifying the experimentInfo whose data fields need to be updated */

string labServerID
/* the GUID identifying the lab server in which the experiment is executed */

string labServerName
/* the name of the lab server in which the experiment is executed */

string labClientVersion
/* the version of the lab client of the experiment*/

string labClientName
/* the name of the lab client of the experiment */

string providerName
/* the name of the provider of the lab server */

string lssID
/* the GUID identifying the lab side scheduling server which manages the reserve information of this experiment */

Returns:

bool modified.

/*true if experimentInfo was successfully modified, ==false otherwise.*/

RemoveExperimentInfo

Purpose:

/*Delete the experiment information specified by the experimentInfoIDs. */

Arguments:

`int[] experimentInfoIDS`
/* An array of experimentInfo IDs specifying the experiments to be removed */

Returns:

`int[] unremovedExperimentIDs`
/*An array of ints containing the IDs of all experiments not successfully removed, i.e., those for which the operation failed. */

ListExperimentInfoIDs

Purpose:

/*Enumerates IDs of all the experimentInfos */

Arguments:

none

Returns:

`int[] experimentInfoIDs`
/* An array of ints containing the IDs of all the experimentInfos */

*/

ListExperimentInfoIDByExperiment

Purpose:

/*enumerates the ID of the information of a particular experiment specified by labClientName and labClientVersion*/

Arguments:

`string labServerName`

/* the name of the lab server in which the experiment is executed */

string labClientVersion
/* the version of the lab client of the experiment*/

Returns:

int experimentInfoID
/* the ID of the requested experiment */

GetExperimentInfos

Purpose:

/*Returns an array of the immutable experimentInfo objects that correspond to the supplied experimentInfo IDs. */

Arguments:

int[] experimentInfoIDs
/* The IDs identifying the experimentInfos whose information is being requested. */

Returns:

UssExperimentInfo[] experimentInfos
/* An array of immutable objects describing the specified ExperimentInfos; if the nth experimentInfoID does not correspond to a valid experimentInfo, the nth entry in the return array will be null.*/

ListLSSURLByExperiment

Purpose:

/*enumerates the url of the LSS which is in charge of a particular experiment specified by labClientName and labClientVersion*/

Arguments:

string labServerName
/* the name of the lab server in which the experiment is executed */

string labClientVersion
/* the version of the lab client of the experiment*/

Returns:

string lssURL
/* the url of requested LSS*/

ListLSSIDByExperiment

Purpose:

/*enumerates the GUID of the LSS which is in charge of a particular experiment specified by labClientName and labClientVersion*/

Arguments:

`string labServerName`
/* the name of the lab server in which the experiment is executed */

`string labClientVersion`
/* the version of the lab client of the experiment*/

Returns:

`string lssURL`
/* the GUID of requested LSS*/

LSSInfo Management Methods

AddLSSInfo

Purpose:

/* Add information of a particular lab side scheduling server identified by lssID */

Arguments:

`string lssID`
/* the GUID identifying the lab side scheduling server */

`string lssName`
/* the name of the lab side scheduling server */

`string lssURL`
/* the URL of the lab side scheduling server*/

Returns:

`int lssInfoID`
/*The unique ID which identifies the LSSInfo added. >0 was successfully added, ==-1 otherwise. */

ModifyLSSInfo

Purpose:

/*Updates the data fields for the LSSInfo specified by the lssInfoID; note lssInfoID may not be changed*/

Arguments:

`int lssInfoID`

/* the ID identifying the lssINfo whose data fields need to be updated*/

string lssID
/* the GUID identifying the lab side scheduling server */

string lssName
/* the name of lab side scheduling server */

string lssURL
/* the url of the lab side scheduling server */

Returns:

bool modified.
/*true if lssInfo was successfully modified, ==false otherwise.*/

RemoveLSSInfo

Purpose:

/*Delete the information of lab side scheduling servers identified by lssInfoIDs */

Arguments:

int[] lssInfoIDs
/* an array of IDs identifying the lab side scheduling servers whose information will be removed */

Returns:

int[] unremovedLSSInfoIDs
/*An array of ints containing the IDs of all LSS informations not successfully removed, i.e., those for which the operation failed. */

ListLSSInfoIDs

Purpose:

/*Enumerates IDs of all the lssInfos */

Arguments:

none

Returns:

int[] lssInfoIDs
/* An array of ints containing the IDs of all the lssInfos */

GetLSSInfos

Purpose:

/*Returns an array of the immutable LSSInfo objects that correspond to the supplied lssInfo IDs. */

Arguments:

int[] lssInfoIDs
/* The IDs identifying the lssInfos whose information is being requested. */

Returns:

```
LSSInfo[] lssInfos
/* An array of immutable objects describing the
specified LssInfos; if the nth lssInfoID does not
correspond to a valid lssInfo, the nth entry in
the return array will be null.*/
```

Credential Set Management Methods

AddCredentialSet

Purpose:

/* Add a credential set of a particular group*/

Arguments:

string serviceBrokerID
/* the GUID identifying the service broker whose domain the group added belongs to*/

string serviceBrokerName
/* the name of the service broker whose domain the group added belongs to*/

string groupName
/* the name of the group added*/

ModifyCredentialSet

Purpose:

/*Updates the data fields for the credential set specified by the credentialSetID; note credentialSetID may not be changed*/

Arguments:

int credentialSetID
/* the ID identifying the credential set whose data fields need to be updated*/

string serviceBrokerID
/* the GUID identifying the service broker whose domain the group added belongs to*/

`string serviceBrokerName`
/* the name of the service broker whose domain the group added belongs to*/

`string groupName`
/* the name of the group added*/

Returns:

`bool modified.`
/*true if credential set was successfully modified, ==false otherwise.*/

RemoveCredentialSet

Purpose:

/*Delete the credential sets specified by the credentialSetIDs. */

Arguments:

`int[] credentialSetIDS`
/* An array of credential set IDs specifying the credential sets to be removed */

Returns:

`int[] unremovedCredentialSetIDs`
/*An array of ints containing the IDs of all credential sets not successfully removed, i.e., those for which the operation failed. */

ListCredentialSetIDs

Purpose:

/*Enumerates IDs of all the credential sets */

Arguments:

`none`

Returns:

`int[] credentialSetIDs`
/* An array of ints containing the IDs of all the credential sets */

GetCredentialSets

Purpose:

/*Returns an array of the immutable credential set objects that correspond to the supplied credential set IDs. */

Arguments:

int[] credentialSetIDs
/* The IDs identifying the credential sets whose information is being requested. */

Returns:

```
UssCredentialSet[] credentialSets
/* An array of immutable objects describing the
specified credential sets; if the nth
credentialSetID does not correspond to a valid
credential set, the nth entry in the return array
will be null.*/
```

Lab-side Scheduling Server

Server Side Scheduling Policy Management Methods

AddLSSPolicy

Purpose:

/*Add lab side scheduling policy to determine whether a reservation from a particular group for a particular experiment should be accepted or not */

Arguments:

int credentialSetID
/* the unique ID identifying credential set of the group to which the lab side scheduling policy applies */

int experimentInfoID
/* the unique ID identifying the experiment to which the lab side scheduling policy applies */

string rule
/* the description of the ussPolicy */

Returns:

int lssPolicyID
/*The unique ID which identifies the lab side scheduling policy added. >0 was successfully added; ==-1 otherwise */

RemoveLSSPolicies

Purpose:

/*Delete the lab side scheduling policies specified by the lssPolicyIDs. */

Arguments:

int[] lssPolicyIDs
/* An array of lab side scheduling policy IDs specifying the policies to be removed*/

Returns:

int[] unremovedPolicyIDs
/*An array of ints containing the IDs of all Policies not successfully removed, i.e., those for which the operation failed. */

ModifyLSSPolicy

Purpose:

/*Updates the data fields for the lab side scheduling policy specified by the lssPolicyID; note lssPolicyID may not be changed; */

Arguments:

int lssPolicyID
/* the ID identifying the lab side scheduling policy whose data is being changed */

int credentialSetID
/* the unique ID identifying the members of new credential set to which the server side scheduling policy applies */

int experimentInfoID
/* the unique ID identifying the new experiment to which the server side scheduling policy applies */

string rule
/* the description of the new lab side scheduling policy*/

Returns:

void none

ListLSSPolicyIDsByExperiment

Purpose:

/*Enumerates all IDs of the lab side scheduling policies for a particular experiment identified by the experimentInfoID */

Arguments:

int experimentInfoID
/* the unique ID identifying the experimet that whose scheduling policies need to be listed */

Returns:

int[] lssPolicyIDs
/* An array of ints containing the IDs of all the lab side Scheduling Policies of specified experiment* /

GetServerSchedulingPolicies

Purpose:

/*Returns an array of the immutable ServerSchedulingPolicy objects that correspond to the supplied lssPolicy IDs. */

Arguments:

int[] lssPolicyIDs
/* The IDs identifying the policies whose information is being requested. */

Returns:

LSSPolicy[] lssPolicies
/* An array of immutable objects describing the specified server scheduling policies; if the nth lssPolicyID does not correspond to a valid policy, the nth entry in the return array will be null.*/

Time block Management Methods

AddTimeBlock

Purpose:

/*Add a time block in which users with a particular credential set are allowed to access a particular lab server */

Arguments:

string labServerID
/* the GUID identifying the lab server which the time block belongs to */

int credentialSetID
/* the unique ID of the credential set identifying the group whose members are allowed to use the lab server*/

DateTime startTime
/* the start time of the time block; note the startTimeo is time in UTC */

DateTime endTime

/* the end time of the time block; note the endTime is time in UTC */

int recurrenceID
/* the ID of the recurrence this time block belongs to */

Returns:

int timeBlockID
/*The unique ID which identifies the time block added. >0 was successfully added; ==-1 otherwise */

RemoveTimeBlocks

Purpose:

/*Delete the time blocks specified by the timeBlockIDs. */

Arguments:

int[] timeBlockIDs
/* An array of time block IDs specifying the time blocks to be removed*/

Returns:

int[] unremovedTimeBlockIDs
/*An array of ints containing the IDs of all time blocks not successfully removed, i.e., those for which the operation failed. */

ModifyTimeBlock

Purpose:

/*Updates the data fields for the time block specified by the timeBlockID; note timeBlockID may not be changed; */

Arguments:

int timeBlockID
/* the ID identifying the time block whose data is being changed */

string labServerID
/* the GUID identifying the new lab server which the time block belongs to */

int credentialSetID
/* the unique ID of the new credential set identifying the group whose members are allowed to use the lab server*/

DateTime startTime
/* the new start time of the time block; note the startTime is the time in UTC */

DateTime endTime
/* the new end time of the time block; note the endTime is the time in UTC*/

Returns:

bool modified
/*true if modified successfully, false otherwise*/

ListTimeBlockIDsByLabServer

Purpose:

/*Enumerates all IDs of the time blocks belonging to a particular lab server identified by the labserverID */

Arguments:

string labServerID
/* the GUID identifying lab server that whose time blocks need to be listed */

Returns:

int[] timeBlockIDs
/* An array of ints containing the IDs of all the time blocks of specified lab server* /

ListTimeBlockIDsByGroup

Purpose:

/*Enumerates all IDs of the time blocks during which the members of a particular group identified by the credentialSetID are allowed to access a particular lab server identified by the labServerID */

Arguments:

string labServerID
/* the GUID identifying lab server that whose time blocks need to be listed */

int credentialSetID
/* the unique ID of credential set identifying a group whose time blocks for the lab server above need to be listed */

Returns:

int[] timeBlockIDs

/* An array of ints containing the IDs of all the time blocks during which the members of a particular group are allowed to access a particular lab server * /

ListTimeBlockIDsByTimeChunk

Purpose:

/*Enumerates the IDs of the time blocks during which the members of a particular group identified by the credentialSetID are allowed to use a particular lab server in a particular time chunk */

Arguments:

`string serviceBrokerID`
/* the GUID identifying the service broker whose domain the group belongs to*/

`string groupName`
/* the name of the group */

`string ussID`
/* the GUID identifying the user side scheduling server which manages the reservation from the members of the group */

`string labServerID`
/* the GUID identifying the lab server */

`string startTime`
/* the start time (UTC) of the time chunk*/

`string endTime`
/* the end time of (UTC) the time chunk */

Returns:

`int[] timeBlockIDs`
/* An array of ints containing the IDs of all the time blocks during which the members of a particular group are allowed to use a particular lab server in a particular time chunk * /

ListTimeIDs

Purpose:

/*Enumerates IDs of the all the time blocks in the LSS*/

Arguments:

`None`

Returns:

`int[] timeBlockIDs`

/* An array of ints containing the IDs of all the time blocks in the LSS* /

GetTimeBlocks

Purpose:

/*Returns an array of the immutable TimeBlock objects that correspond to the supplied time block IDs. */

Arguments:

int[] timeBlockIDs
/* The IDs identifying the time blocks whose information is being requested. */

Returns:

TimeBlock[] timeBlocks
/* An array of immutable objects describing the specified time blocks; if the nth timeBlockID does not correspond to a valid time block, the nth entry in the return array will be null.*/

Experiment Information Management Methods

AddExperimentInfo

Purpose:

/* Add information of a particular experiment*/

Arguments:

string labServerID
/* the GUID identifying the lab server in which the experiment is executed */

string labServerName
/* the name of the lab server in which the experiment is executed */

string labClientVersion
/* the version of the lab client of the experiment*/

string labClientName
/* the name of the lab client of the experiment */

string providerName
/* the name of the provider of the lab server */

int quantum
/* the maximum divisor of the experiment's possible execution time; note the unite of quantum is minute */

int preParеTime
/* the start up time needed before the execution of the experiment; note the unite of start up time is minute*/

int recoverTime
/* the cool down time needed after the execution of the experiment; note the unite of cool down time is minute*/

int minimumTime
/* the experiment's minimum execution time */

int earlyArriveTime
/* the time users are allowed to arrive earlier than his reservation */

Returns:

int experimentInfoID
/*The unique ID which identifies the experiment information added. >0 was successfully added; ==-1 otherwise */

RemoveExperimentInfo

Purpose:

/*Delete the experiment information specified by the experimentInfoIDs. */

Arguments:

int[] experimentInfoIDs
/* An array of experimentInfoIDs specifying the experiment information to be removed*/

Returns:

int[] unremovedExperimentInfoIDs
/*An array of ints containing the IDs of all experiment information not successfully removed, i.e., those for which the operation failed. */

ModifyExperimentInfo

Purpose:

/*Updates the data fields for the experiment information specified by the experimentInfoID; note experimentInfoID may not be changed; */

Arguments:

int experimentInfoID
/* the ID identifying the experiment information whose data is being changed */

string labServerID

/* the GUID of the lab server in which the experiment is executed */

string labServerName
/* the new name of the lab server in which the experiment is executed */

string labClientVersion
/*the version of the lab client through which the experiment can be executed*/

string labClientName
/*the name of the lab client through which the experiment can be executed*/

string providerName
/* the new name of the provider of the lab server */

int quantum
/* the new maximum divisor of the experiment's possible execution time; note the unite of quantum is minute */

int prepareTime
/* the new start up time needed before the execution of the experiment; note the unite of start up time is minute*/

int recoverTime
/* the new cool down time needed after the execution of the experiment; note the unite of cool down time is minute*/

int minimumTime
/* the new experiment's minimum execution time */

int earlyArriveTime
/* the time users are allowed to arrive earlier than his reservation */

Returns:

bool modified
/*true if modified successfully, false otherwise*/

ListExperimentInfoIDsByLabServer

Purpose:

/*Enumerates IDs of the information of all the experiments belonging to certain lab server identified by the labserverID */

Arguments:

string labServerID

/* the GUID identifying lab server whose experiments' information need to be listed */

Returns:

int[]experimentInfoIDs
/* An array of ints containing the IDs of the information of all the experiments belonging to specified lab server* /

RetriveLabServerName

Purpose:

/* get the labserver name according to the labserver ID */

Arguments:

string labServerID
/* the GUID identifying lab server whose experiments' information need to be listed */

Returns:

String labServerName
/* the name of the lab server with the particular lab server ID* /

ListExperimentInfoIDsByExperiment

Purpose:

/*Enumerates the ID of the information of a particular experiment specified by labClientName and labClientVersion*/

Arguments:

string labClientName
/* the Name of the client for an experiment that whose information need to be listed */

string labClientVersion
/* the version of the client for an experiment that whose information need to be listed */

Returns:

int experimentInfoID
/* the ID of the information of a particular experiment. -1 if such a experiment info can not be retrieved * /

ListExperimentInfoIDs

Purpose:

/*Enumerates IDs of the information of all the experiments in the LSS*/

Arguments:

None

Returns:

int[]experimentInfoIDs
/* An array of ints containing the IDs of the information of all the experiments in the LSS* /

GetExperimentInfos

Purpose:

/*Returns an array of the immutable ExperimentInfo objects that correspond to the supplied experiment information IDs. */

Arguments:

int[] experimentInfoIDs
/* The IDs identifying the experiment information being requested. */

Returns:

LssExperimentInfo[] experimentInfos
/* An array of immutable objects describing the specified experiment information; if the nth experimentInfoID does not correspond to a valid experiment scheduling property, the nth entry in the return array will be null.*/

USSInfo Management Methods

AddUSSInfo

Purpose:

/* Add information of a particular user side scheduling server identified by ussID */

Arguments:

string ussID
/* the GUID identifying the user side scheduling server */

string ussName
/* the name of the user side scheduling server */

string ussURL
/* the URL of the user side scheduling server*/

Returns:

int ussInfoID

/*The unique ID which identifies the ussInfo added. >0 was successfully added; ==-1 otherwise */

ModifyUSSInfo

Purpose:

/*Updates the data fields for the USS information specified by the ussInfoID; note ussInfoID may not be changed */

Arguments:

`int ussInfoID`
/*the unique ID identifying the ussInfo whose date fields need to be updated*/

`string ussID`
/*The GUID identifying the user side scheduling server */

`string ussName`
/* the name of the user side scheduling server */

`string ussURL`
/* the URL of the user side scheduling server*/

Returns:

`bool modified`
/*true if modified successfully, false otherwise*/

RemoveUSSInfo

Purpose:

/*Delete the uss information specified by the ussInfoIDs. */

Arguments:

`int[] ussInfoIDs`
/* An array of USS information IDs specifying the USS information to be removed*/

Returns:

`int[] unremovedUSSInfoIDs`
/*An array of ints containing the IDs of all USS information not successfully removed, i.e., those for which the operation failed. */

ListUSSInfoIDs

Purpose:

/*Enumerates the IDs of the information of all the USS */

Arguments:

`None`

Returns:

`int[] ussInfoIDs`
/* the array of ints contains the IDs of the information of all the USS * /

ListUSSInfoID

Purpose:

/*Enumerates the ID of the information of a particular USS specified by ussID*/

Arguments:

`string ussID`
/* the GUID identifying USS that whose information need to be listed */

Returns:

`int ussInfoID`
/* the ID of the information of a particular USS, -1 if such a ussInfo can not be retrieved * /

GetUSSInfos

Purpose:

/*Returns an array of the immutable USSInfo objects that correspond to the supplied USS information IDs. */

Arguments:

`int[] ussInfoIDs`
/* The IDs identifying the USS information being requested. */

Returns:

`USSInfo[] ussInfos`
/* An array of immutable objects describing the specified USS information; if the nth ussInfoID does not correspond to a valid experiment scheduling property, the nth entry in the return array will be null.*/

Permitted Experiments Management Methods

AddPermittedExperiment

Purpose:

/* Add permission of a particular experiment being executed in a particular time block */

Arguments:

int experimentInfoID
/* the unique ID identifying the experiment which is given the permission to a timeblock */

int recurrenceID
/* the ID identifying the recurrence in which the experiment is permitted to be executed */

Returns:

int permittedExperimentID
/*The unique ID which identifies the permission added. >0 was successfully added; ==-1 otherwise */

RemovePermittedExperiments

Purpose:

/*Delete permissions of a particular experiment being executed in a particular time block. */

Arguments:

int[] permittedExperimentIDs
/* An array of permittedExperimentIDs specifying the permissions to be removed*/

Returns:

int[] unremovedPermittedExperimentIDs
/*An array of ints containing the IDs of all permissions not successfully removed, i.e., those for which the operation failed. */

ListPermittedExperimentInfoIDsByTimeBlock

Purpose:

/*Enumerates the IDs of the information of the permitted experiments for a particular time block identified by the timeBlockID*/

Arguments:

int timeBlockID
/* the unique ID identifying the timeblock whose permitted experiments need to be listed */

Returns:

int[] permittedExperimentInfoIDs
/* An array of ints containing the IDs of the information of the permitted experiments for a particular time block identified by the timeBlockID * /

GetPermittedExperiments

Purpose:

/*Returns an array of the immutable PermittedExperiment objects that correspond to the supplied permittedExperiment IDs. */

Arguments:

`int[] permittedExperimentIDs`
/* The IDs identifying the USS permission being requested. */

Returns:

`PermittedExperiment[] permisttedExperiments`
/* An array of immutable objects describing the specified PermittedExperiments; if the nth permittedExperimentID does not correspond to a valid permitted experiment, the nth entry in the return array will be null.*/

ListPermittedExperimentID

Purpose:

/* retrieve unique ID of the PerimmttiedExperiment which represents the permission of executing a particular experiment in a particular time block
*/

Arguments:

`int experimentInfoID`
/* the ID of the experiment Informaiton */

`int timeBlockID`
/* the unique ID identifying the timeblock */

Returns:

`int permittedExperimentID`
/* unique ID of the PerimmttiedExperiment which represents the permission of executing a particular experiment in a particular time block */

ListPermittedExperimentIDByRecur

Purpose:

/* retrieve unique ID of the PerimmttiedExperiment which represents the permission of executing a particular experiment in a particular recurrence*/

Arguments:

`int experimentInfoID`
/* the ID of the experiment Informaiton */

int recurrenceID
/* the unique ID identifying the recurrence */

Returns:

int permittedExperimentID
/* the unique ID of the PerimmttiedExperiment which represents the permission of executing a particular experiment in a particular recurrence */

ListPermittedExperimentInfoIDByRecurrence

Purpose:

/* enumerates the IDs of information of the permitted experiments for a particular recurrence identified by the recurrenceID */

Arguments:

int recurrenceID
/* the unique ID identifying the recurrence */

Returns:

Int[] permittedExperimentIDs
/* the IDs of information of the permitted experiments for a particular recurrence identified by the recurrenceID */

CheckPermission

Purpose:

/*Determing whether the specified experiment has the permission of being executed in the specified time block */

Arguments:

int experimentInfoID
/* The IDs identifying the experiment. */

int timeBlockID
/* The IDs identifying the time block. */

Returns:

bool isPermitted
/*true if the specified experiment has the permission of being executed in the specified time block; false otherwise*/

Reservation Information Management Methods

AddReservationInfo

Purpose:

/* add reservation information. */

Arguments:

`string serviceBrokerID`
/* the ID of the service broker where the reservation from*/

`string groupName`
/* the name of the group that the user who made the reservation belongs to*/

`string ussID`
/* the ID of the USS which manage this reservation in the user side*/

`string labClientName`
/* the name of the lab client */

`string labClientVersion`
/* the version of the lab client*/

`DateTime startTime`
/* the start time of the reservation (UTC)*/

`DateTime endTime`
/* the end time of the reservation (UTC) */

Returns:

`int  reservationInfoID`
/* the unique ID identifying the reservation information added, >0 successfully added, -1 otherwise */

AddReservationInfo

Purpose:

/* add reservation information. */

Arguments:

`int credentialSetID`
/* the ID of the credential set which the user who made the reservation has*/

`int experimentInfoID`
/* the ID of the experiment inforomation */

`DateTime startTime`
/* the start time of the reservation (UTC)*/

`DateTime endTime`
/* the end time of the reservation (UTC) */

Returns:

int reservationInfoID
/* the unique ID identifying the reservation information added, >0 successfully added, -1 otherwise */

RemoveReservationInfoByIDs

Purpose:

/*Delete the reservation information specified by the reservationInfoIDs. */

Arguments:

int[] reservationInfoIDs
/* An array of reservarion information IDs specifying the reservation information to be removed*/

Returns:

int[] unremovedReservationInfoIDs
/*An array of ints containing the IDs of all reservation information not successfully removed, i.e., those for which the operation failed. */

RemoveReservationInfo

Purpose:

/* remove the reservation information. */

Arguments:

string serviceBrokerID
/* the ID of the service broker where the reservation from*/

string groupName
/* the name of the group that the user who made the reservation belongs to*/

string ussID
/* the ID of the USS which manage this reservation in the user side*/

string labClientName
/* the name of the lab client */

string labClientVersion
/* the version of the lab client*/

DateTime startTime
/* the start time of the reservation (UTC)*/

DateTime endTime

/* the end time of the reservation (UTC) */

Returns:

boolean removed
/* true remove successfully, false otherwise*/

ListReservationInfoIDsByExperiment

Purpose:

/*Enumerates all IDs of the reservations made to a particular experiment identified by the experimentInfoID */

Arguments:

int experimentInfoID
/* the ID identifying the experiment which reservations to be listed are made to*/

Returns:

int[] reservationInfoIDs
/* An array of ints containing the IDs of all the reservation information made to the specified experiment */

ListReservationInfoIDs

Purpose:

/* enumerates all IDs of the reservations made to a particular experiment from a particular group between the start time and the end time. It also includes the reservation, part of it is in the checked period */

Arguments:

string serviceBrokerID
/* the ID of the service broker where the reservation from*/

string groupName
/* the name of the group that the user who made the reservation belongs to*/

string ussID
/* the ID of the USS which manage this reservation in the user side*/

string labClientName
/* the name of the lab client */

string labClientVersion
/* the version of the lab client*/

DateTime startTime
/* the start time of the checked period(UTC)*/

DateTime endTime
/* the end time of the checked period (UTC) */

Returns:

Int[] reservationIDs
/* all IDs of the reservations made to a particular experiment from a particular group between the start time and the end time.*/

ListReservationInfoIDsByLabServer

Purpose:

/* retrieve reservation made to a particular labserver during a given time chunk.*/

Arguments:

string labServerID
/* the id identifying the lab server*/

DateTime startTime
/* the start time of the checked period(UTC)*/

DateTime endTime
/* the end time of the checked period (UTC) */

Returns:

Int[] reservationIDs
/* the ID of the reservations made to a particular labserver during a given time chunk.*/

ListReservationInfoIDs

Purpose:

/* to select reservation Infos accorrding to given criterion*/

Arguments:

string labServerID
/* the ID of the lab server*/

int experimentInfoID
/* the ID of the experiment information the reservation is to execute*/

int credentialID
/* the ID of the credential set that the user who made the reservation belongs to*/

DateTime timeAfter
/* the start time of the checked period(UTC)*/

DateTime timeBefore
/* the end time of the checked period (UTC) */

Returns:

Int[] reservationIDs
/* all IDs of the selected reservation Infos accorrding to given criterion
*/

GetReservationInfos

Purpose:

/*Returns an array of the immutable ReservationInfo objects that correspond to the supplied reservationInfoIDs. */

Arguments:

int[] reservationInfoIDs
/* The IDs identifying the reservations whose information is being requested. */

Returns:

ReservationInfo[] reservationInfo
/* An array of immutable objects describing the specified reservations; if the nth reservationInfoID does not correspond to a valid reservation information, the nth entry in the return array will be null.*/

Credential Set Management Methods

AddCredentialSet

Purpose:

/* Add a credential set of a particular group*/

Arguments:

string serviceBrokerID
/* the GUID identifying the service broker whose domain the group added belongs to*/

string serviceBrokerName
/* the name of the service broker whose domain the group added belongs to*/

string groupName
/* the name of the group added*/

string ussID
/* the GUID identifying the user side scheduling server which manages the reservation from the members of the group added */

Returns:

int credentialSetID
/*The unique ID which identifies the credential set added. >0 was successfully added; ==-1 otherwise */

ModifyCredentialSet

Purpose:

/*Updates the data fields for the credential set specified by the credentialSetID; note credentialSetID may not be changed */

Arguments:

int credentialSetID
/*the unique ID identifying the credential set whose date fields need to be updated*/

string serviceBrokerID
/* the GUID identifying the service broker whose domain the group updated belongs to*/

string serviceBrokerName
/* the name of the service broker whose domain the group updated belongs to*/

string groupName
/* the name of the group updated*/

string ussID
/* the GUID identifying the user side scheduling server which manages the reservation from the members of the group updated */

Returns:

bool modified
/*true if modified successfully, false otherwise*/

RemoveCredentialSets

Purpose:

/* remove a credential set specified by the credentialsetsIDS*/

Arguments:

int[] credentialSetIDs
/* An array of credentialset IDs specifying the credential sets to be removed*/

Returns:

int[] unremovedCredentialSetIDs
/*An array of ints containing the IDs of all credential sets not successfully removed, i.e., those for which the operation failed. */

ListCredentialSetIDs

Purpose:

/*Enumerates the IDs of the information of all the credential set */

Arguments:

None

Returns:

int[] credentialSetIDs
/* the array of ints contains the IDs of all the credential set* /

GetCredentialSets

Purpose:

/*Returns an array of the immutable Credential objects that correspond to the supplied credentialSet IDs. */

Arguments:

int[] credentialSetIDs
/* The IDs identifying the credentialSet being requested. */

Returns:

LssCredentialSet[] credentialSets
/* An array of immutable objects describing the specified Credential Set information; if the nth credentialSetID does not correspond to a valid experiment scheduling property, the nth entry in the return array will be null.*/

Recurrence Management Methods

AddRecurrence

Purpose:

/*Add recurrence*/

Arguments:

DateTime recurrenceStartDate
/* the start date of the recurrence UTC */

DateTime recurrenceEndDate
/* the end date of the recurrence UTC*/

String recurrenceType
/* the type of recurrence : none, weekly,daily,*/

TimeSpan recurrenceStartTime
/* the start time of the day of the recurrence expressed in timespan*/

TimeSpan recurrenceEndTime
/* the end time of the day of the recurrence expressed in timespan*/

string labServerID
/* the ID of the lab server this recurrence assigned to*/

int credentialSetID
/* the ID of the credential set this recurrence assigned to */

Returns:

int recurrenceID
/* the uniqueID which identifies the recurrence added, >0 was successfully added; ==-1 otherwise
*/

RemoveRecurrence

Purpose:

/*Delete the recurrences specified by the recurrenceIDs. */

Arguments:

int[] recurrenceIDs
/* An array of recurrence IDs specifying the time blocks to be removed*/

Returns:

int[] unremovedRecurrenceIDs
/*An array of ints containing the IDs of all recurrences not successfully removed, i.e., those for which the operation failed. */

GetRecurrence

Purpose:

/*Returns an array of the immutable Recurrence that correspond to the supplied recurrenceIDs. */

Arguments:

int[] recurrenceIDs
/* The IDs identifying the recurrence being requested. */

Returns:

Recurrence[] recurrences
/* An array of immutable objects describing the specified Recurrence information; if the nth recurrernceID does not correspond to a valid experiment scheduling property, the nth entry in the return array will be null.*/

ListRecurrenceIDs

Purpose:

/*Enumerates the IDs of the information of all the recurrences */

Arguments:

None

Returns:

int[] recurrenceIDs
/* the array of ints contains the IDs of all the recurrences* /

ListRecurrenceIDsByLabServer

Purpose:

/* enumerates all IDs of the recurrences belonging to a particular lab server identified by the labserverID */

Arguments:

String labServerID
/* the ID of the lab server*/

Returns:

int[] recurrenceIDs
/* all IDs of the recurrences belonging to a particular lab server identified by the labserverID * /

Other API

RetrieveAvailableTimePeriods

Purpose:

/*Retrieve all the available time for a given experiment and credential set within a particular USS specified interval */

Arguments:

string serviceBrokerID
/* the GUID identifying the service broker which is one of the properties of the credential set*/

string groupName
/* the name of the group which is one of the properties of the credential set */

string ussID
/* the GUID identifying the user side scheduling server which is one of the properties of the credential set */

string labClientName
/*the name of the client of the experiment whose available time is requested*/

string labClientVersion
/*the version of the client of the experiment whose available time is requested*/

DateTime startTime
/* the start Time of time the particular USS specified interval */

DateTime endTime
/* the end Time of time the particular USS specified interval */

Returns:

ArrayList availableTimePeriods
/* arrayList containing available time periods time for a given experiment and credential set within a particular USS specified interval */

ConfirmReservation

Purpose:

/* Returns an Boolean indicating whether a particular reservation from a USS is confirmed and added to the database in LSS successfully. If it fails, exception will be throw out indicating he reason for rejection.*/

Arguments:

String serviceBrokerID

/* the ID of the service broker which manage the user's information */

String groupName
/* the name of the group the user belongs to*/

String ussID
/* the GUID of the USS that manage this user's reservation for the experiment*/

String labClientName
/* the name of the lab client which runs the experiment*/

String labClientVersion
/* the version of the lab client*/

DateTime startTime
/* the start time of the reservation (the local time of the LSS)*/

DateTime endTime
/* the end time of the reservation (the local time of the LSS)*/

Returns:

String confirmation
/* the notification whether the reservation is confirmed. If not, notification will give a reason */

RetriveTimeSlots

Purpose:

/* given a time period defined by the start time and the end tiime, return the time slots defined by the quatum of the experiment during this time period*/

Arguments:

String labClientName
/* the name of the lab client which runs the experiment*/

String labClientVersion
/* the version of the lab client*/

DateTime startTime
/* the start time of the time period (the local time of the LSS)*/

DateTime endTime
/* the end time of the time period (the local time of the LSS)*/

Returns:

TimePeriod[] timeslots

/* given a time period defined by the start time and the end tiime, return the time slots defined by the quatum of the experiment during this time period */

Web Service Methods

User-side Scheduling Server

RedeemReservation

Purpose:

/*Returns an Boolean indicating whether it the right time for a particular user to execute a particular experiment*/

Arguments:

string username

/*The name of the user who is redeeming reservation*/

string serviceBrokerID

/*The service broker ID whose domain the user belongs to*/

string labClientName

/*The name of the lab client , the reservation on which is redeeming*/

string labClientVersion

/*The version of the lab client*/

Returns:

bool redeemed

/*true if reservation was redeemed, in order for this happen, the current time need to be covered by the time period defined by the startTime and endTime of the reservation; ==false otherwise.*/

RevokeReservation

Purpose:

/*remove all the reservation for certain lab server being covered by the revocation time and send emails to the affected staff and users */

Arguments:

string labServerID

/* the ID identifying the lab server whose time is being revoked */

DateTime startTime
/* The start time of the revocation period. */

DateTime endTime
/* The end time of the revocation period. */

Returns:

void none

AddCredentialSet

Purpose:

/* Add a credential set of a particular group*/

Arguments:

string serviceBrokerID
/* the GUID identifying the service broker whose domain the group added belongs to*/

string serviceBrokerName
/* the name of the service broker whose domain the group added belongs to*/

string groupName
/* the name of the group added*/

RemoveCredentialSet

Purpose:

/*Delete the credential sets specified by the credentialSetIDs. */

Arguments:

int[] credentialSetIDS
/* An array of credential set IDs specifying the credential sets to be removed */

Returns:

int[] unremovedCredentialSetIDs
/*An array of ints containing the IDs of all credential sets not successfully removed, i.e., those for which the operation failed. */

AddExperimentInfo

Purpose:

/* Add information of a particular experiment*/

Arguments:

string labServerID

/* the GUID identifying the lab server in which the experiment is executed */

string labServerName
/* the name of the lab server in which the experiment is executed */

string labClientVersion
/* the version of the lab client of the experiment*/

string labClientName
/* the name of the lab client of the experiment */

string providerName
/* the name of the provider of the lab server */

string lssID
/* the GUID identifying the lab side scheduling server which manages the reserve information of this experiment */

Returns:

int experimentInfoID
/*The unique ID which identifies the experiment added. >0 was successfully added, ==-1 otherwise. */

AddLSSInfo

Purpose:

/* Add information of a particular lab side scheduling server identified by lssID */

Arguments:

string lssID
/* the GUID identifying the lab side scheduling server */

string lssName
/* the name of the lab side scheduling server */

string lssURL
/* the URL of the lab side scheduling server*/

Returns:

int lssInfoID
/*The unique ID which identifies the LSSInfo added. >0 was successfully added, ==-1 otherwise. */

Lab-side Scheduling Server

ConfirmReservation

Purpose:

/*Returns a notification indicating whether a particular reservation from a USS is confirmed and added to the database in LSS successfully. If it fails, the notification will indicate
the reason for rejection.*/

Arguments:

`string serviceBrokerID`
/* the GUID identifying the service broker whose domain the group whose member made the reservation requested belongs to */

`string groupName`
/* the name of the group whose member made the reservation requested*/

`string ussID`
/* the GUID identifying the user side scheduling server which the reservation is requested from*/

`string labClientName`
/*the name of the client of the experiment whose time is requested to be reserved*/

`string labClientVersion`
/*the version of the client of the experiment whose time is requested to be reserved*/

`DateTime startTime`
/*the startTime of the reservation requested. note startTime is the time in UTC */

`DateTime endTime`
/* the endTime of the reservation requested. note endTime is the time in UTC */

Returns:

`String notification`
/* if validated and successfully added to the database in LSS, true; otherwise, false. Before the reservation being added to the database of LSS, LSS needs to judge whether the reservation information is confirmed. To validate the confirmation, all the followings must be satisfied.

1. The reservation is in the current available time periods for the group that the reservation comes from

2. All the corresponding lab server side policies which the reservation comes from should be satisfied.

3. All the scheduling properties for the experiment which the reservation is made to should be satisfied. */

RemoveReservationInfo

Purpose:

/*Remove reservation information. */

Arguments:

string serviceBrokerID
/* the GUID identifying the service broker whose domain the group whose member made the reservation removed belongs to */

string groupName
/* the name of the group whose member made the reservation removed*/

string ussID
/* the GUID identifying the user side scheduling server which the reservation removed is requested from */

string labClientName
/*the name of the client of the experiment whose time is reserved*/

string labClientVersion
/*the version of the client of the experiment whose time is reserved*/

DateTime startTime
/*the startTime of the reservation removed. note startTime is the time in UTC */

DateTime endTime
/* the endTime of the reservation removed. note endTime is the time in UTC */

Returns:

bool removed
/* true was successfully removed; false otherwise */

RetrieveAvailableTimePeriods

Purpose:

/*Retrieve all the available time for a given experiment and credential set within a particular USS specified interval */

Arguments:

string serviceBrokerID
/* the GUID identifying the service broker which is one of the properties of the credential set*/

string groupName
/* the name of the group which is one of the properties of the credential set */

string ussID
/* the GUID identifying the user side scheduling server which is one of the properties of the credential set */

string labClientName
/*the name of the client of the experiment whose available time is requested*/

string labClientVersion
/*the version of the client of the experiment whose available time is requested*/

DateTime startTime
/* the start Time of time the particular USS specified interval */

DateTime endTime
/* the end Time of time the particular USS specified interval */

Returns:

ArrayList availableTimePeriods
/* arrayList containing available time periods time for a given experiment and credential set within a particular USS specified interval */

AddUSSInfo

Purpose:

/* Add information of a particular user side scheduling server identified by ussID */

Arguments:

string ussID
/* the GUID identifying the user side scheduling server */

string ussName
/* the name of the user side scheduling server */

string ussURL
/* the URL of the user side scheduling server*/

Returns:

int ussInfoID

/*The unique ID which identifies the ussInfo added. >0 was successfully added; ==-1 otherwise */

AddCredentialSet

Purpose:

/* Add a credential set of a particular group*/

Arguments:

`string serviceBrokerID`
/* the GUID identifying the service broker whose domain the group added belongs to*/

`string serviceBrokerName`
/* the name of the service broker whose domain the group added belongs to*/

`string groupName`
/* the name of the group added*/

Returns:

`Bool added`
/* true if the credential sets are added successfully, false, otherwise */

RemoveCredentialSet

Purpose:

/* Remove a credential set of a particular group*/

Arguments:

`string serviceBrokerID`
/* the GUID identifying the service broker whose domain the group added belongs to*/

`string serviceBrokerName`
/* the name of the service broker whose domain the group removed belongs to*/

`string groupName`
/* the name of the group removed*/

Returns:

`Bool added`
/* true if the credential sets are removed successfully, false, otherwise */

RetriveTimeSlots

Purpose:

/* given a time period defined by the start time and the end tiime, return the time slots defined by the quatum of the experiment during this time period*/

Arguments:

`String labClientName`
/* the name of the lab client which runs the experiment*/

`String labClientVersion`
/* the version of the lab client*/

`DateTime startTime`
/* the start time of the time period (the local time of the LSS)*/

`DateTime endTime`
/* the end time of the time period (the local time of the LSS)*/

Returns:

`TimePeriod[] timeslots`
/* given a time period defined by the start time and the end tiime, return the time slots defined by the quatum of the experiment during this time period */