

Security

Return Types

Constants Summary

Number	#ACCESSIBLE	Constant representing the accessible flag for form security.
Number	#DELETE	Constant representing the delete flag for table security.
Number	#INSERT	Constant representing the insert flag for table security.
Number	#READ	Constant representing the read flag for table security.
Number	#TRACKING	Constant representing the tracking flag for table security (tracks sql insert/update/delete).
Number	#TRACKING_VIEWS	Constant representing the tracking flag for table security (tracks sql select).
Number	#UPDATE	Constant representing the update flag for table security.
Number	#VIEWABLE	Constant representing the viewable flag for form security.

Method Summary

Boolean	#addUserToGroup(a_userUID, groupName)	Adds an user to a named group.
Object	#authenticate(authenticator_solution, method)	Authenticate to the Servoy Server using one of the installed authenticators or the Servoy default authenticator.
Object	#authenticate(authenticator_solution, method, credentials)	Authenticate to the Servoy Server using one of the installed authenticators or the Servoy default authenticator.
Boolean	#canDelete(dataSource)	Returns a boolean value for security rights.
Boolean	#canInsert(dataSource)	Returns a boolean value for security rights.
Boolean	#canRead(dataSource)	Returns a boolean value for security rights.
Boolean	#canUpdate(dataSource)	Returns a boolean value for security rights.
Boolean	#changeGroupName(oldGroupName, newGroupName)	Changes the groupname of a group.
Boolean	#changeUserName(a_userUID, username)	Changes the username of the specified userUID.
Boolean	#checkPassword(a_userUID, password)	Returns true if the password for that userUID is correct, else false.
String	#createGroup(groupName)	Creates a group, returns the groupname (or null when group couldn't be created).
Object	#createUser(username, password)	Creates a new user, returns new uid (or null when group couldn't be created or user already exist).
Object	#createUser(username, password, userUID)	Creates a new user, returns new uid (or null when group couldn't be created or user already exist).
Boolean	#deleteGroup(groupName)	Deletes a group, returns true if no error was reported.
Boolean	#deleteUser(userUID)	Deletes an user.
String	#getClientID()	Returns the client ID.
JSDataSet	#getElementUIDs(formname)	Returns the form elements UUID's as dataset, the one with no name is the form itself.
JSDataSet	#getGroups()	Get all the groups (returns a dataset).
String	#getSystemUserName()	Retrieves the username of the currently logged in user on operating system level.
JSDataSet	#getUserGroups()	Get all the groups of the current user.

JSDataSet [#getUserGroups\(userID\)](#)
Get all the groups for given user UID.

String [#getUserName\(\)](#)
Get the current user name (null if not logged in), finds the user name for given user UID if passed as parameter.

String [#getUserName\(userID\)](#)
Get the current user name (null if not logged in), finds the user name for given user UID if passed as parameter.

String [#getUserUID\(\)](#)
Get the current user UID (null if not logged in); finds the userID for given user_name if passed as parameter.

String [#getUserUID\(username\)](#)
Get the current user UID (null if not logged in); finds the userID for given user_name if passed as parameter.

JSDataSet [#getUsers\(\)](#)
Get all the users in the security settings (returns a dataset).

JSDataSet [#getUsers\(groupName\)](#)
Get all the users in the security settings (returns a dataset).

Boolean [#isUserMemberOfGroup\(groupName\)](#)
Check whatever the current user is part of the specified group

Boolean [#isUserMemberOfGroup\(groupName, userID\)](#)
Check whatever the user specified as parameter is part of the specified group.

Boolean [#login\(username, a_userUID, groups\)](#)
Login to be able to leave the solution loginForm.

void [#logout\(\)](#)
Logout the current user and close the solution, if the solution requires authentication and user is logged in.

void [#logout\(solutionToLoad\)](#)
Logout the current user and close the solution, if the solution requires authentication and user is logged in.

void [#logout\(solutionToLoad, method\)](#)
Logout the current user and close the solution, if the solution requires authentication and user is logged in.

void [#logout\(solutionToLoad, method, argument\)](#)
Logout the current user and close the solution, if the solution requires authentication and user is logged in.

Boolean [#removeUserFromGroup\(a_userUID, groupName\)](#)
Removes an user from a group.

Boolean [#setPassword\(a_userUID, password\)](#)
Set a new password for the given userID.

void [#setSecuritySettings\(dataset\)](#)
Sets the security settings; the entries contained in the given dataset will override those contained in the current security settings.

Boolean [#setUserID\(a_userUID, newUserID\)](#)
Set a new userID for the given userID.

Constants Details

ACCESSIBLE

Constant representing the accessible flag for form security.

Returns

Number

Sample

```
var colNames = new Array();
colNames[0] = 'uuid';
colNames[1] = 'flags';
var dataset = databaseManager.createEmptyDataSet(0,colNames);

var row = new Array();
row[0] = '413a4d69-becb-4ae4-8fdd-980755d6a7fb'; //normally retrieved via security.getElementUUIDs(...)
row[1] = JSSecurity.VIEWABLE|JSSecurity.ACCESSIBLE; // use bitwise 'or' for both
dataset.addRow(row); //setting element security

row = new Array();
row[0] = 'example_data.orders';
row[1] = JSSecurity.READ|JSSecurity.INSERT|JSSecurity.UPDATE|JSSecurity.DELETE|JSSecurity.TRACKING; //use
bitwise 'or' for multiple flags
dataset.addRow(row); //setting table security

security.setSecuritySettings(dataset); //to be called in solution startup method
```

DELETE

Constant representing the delete flag for table security.

Returns

Number

Sample

```
var colNames = new Array();
colNames[0] = 'uuid';
colNames[1] = 'flags';
var dataset = databaseManager.createEmptyDataSet(0,colNames);

var row = new Array();
row[0] = '413a4d69-becb-4ae4-8fdd-980755d6a7fb';//normally retrieved via security.getElementUUIDs(...)
row[1] = JSSecurity.VIEWABLE|JSSecurity.ACCESSIBLE; // use bitwise 'or' for both
dataset.addRow(row);//setting element security

row = new Array();
row[0] = 'example_data.orders';
row[1] = JSSecurity.READ|JSSecurity.INSERT|JSSecurity.UPDATE|JSSecurity.DELETE|JSSecurity.TRACKING; //use
bitwise 'or' for multiple flags
dataset.addRow(row);//setting table security

security.setSecuritySettings(dataset);//to be called in solution startup method
```

INSERT

Constant representing the insert flag for table security.

Returns

Number

Sample

```
var colNames = new Array();
colNames[0] = 'uuid';
colNames[1] = 'flags';
var dataset = databaseManager.createEmptyDataSet(0,colNames);

var row = new Array();
row[0] = '413a4d69-becb-4ae4-8fdd-980755d6a7fb';//normally retrieved via security.getElementUUIDs(...)
row[1] = JSSecurity.VIEWABLE|JSSecurity.ACCESSIBLE; // use bitwise 'or' for both
dataset.addRow(row);//setting element security

row = new Array();
row[0] = 'example_data.orders';
row[1] = JSSecurity.READ|JSSecurity.INSERT|JSSecurity.UPDATE|JSSecurity.DELETE|JSSecurity.TRACKING; //use
bitwise 'or' for multiple flags
dataset.addRow(row);//setting table security

security.setSecuritySettings(dataset);//to be called in solution startup method
```

READ

Constant representing the read flag for table security.

Returns

Number

Sample

```
var colNames = new Array();
colNames[0] = 'uuid';
colNames[1] = 'flags';
var dataset = databaseManager.createEmptyDataSet(0,colNames);

var row = new Array();
row[0] = '413a4d69-becb-4ae4-8fdd-980755d6a7fb';//normally retrieved via security.getElementUUIDs(...)
row[1] = JSSecurity.VIEWABLE|JSSecurity.ACCESSIBLE; // use bitwise 'or' for both
dataset.addRow(row);//setting element security

row = new Array();
row[0] = 'example_data.orders';
row[1] = JSSecurity.READ|JSSecurity.INSERT|JSSecurity.UPDATE|JSSecurity.DELETE|JSSecurity.TRACKING; //use
bitwise 'or' for multiple flags
dataset.addRow(row);//setting table security

security.setSecuritySettings(dataset);//to be called in solution startup method
```

TRACKING

Constant representing the tracking flag for table security (tracks sql insert/update/delete).

Returns

Number

Sample

```
var colNames = new Array();
colNames[0] = 'uuid';
colNames[1] = 'flags';
var dataset = databaseManager.createEmptyDataSet(0,colNames);

var row = new Array();
row[0] = '413a4d69-becb-4ae4-8fdd-980755d6a7fb';//normally retrieved via security.getElementUUIDs(...)
row[1] = JSSecurity.VIEWABLE|JSSecurity.ACCESSIBLE; // use bitwise 'or' for both
dataset.addRow(row);//setting element security

row = new Array();
row[0] = 'example_data.orders';
row[1] = JSSecurity.READ|JSSecurity.INSERT|JSSecurity.UPDATE|JSSecurity.DELETE|JSSecurity.TRACKING; //use
bitwise 'or' for multiple flags
dataset.addRow(row);//setting table security

security.setSecuritySettings(dataset);//to be called in solution startup method
```

TRACKING_VIEWS

Constant representing the tracking flag for table security (tracks sql select).

Returns

Number

Sample

```
var colNames = new Array();
colNames[0] = 'uuid';
colNames[1] = 'flags';
var dataset = databaseManager.createEmptyDataSet(0,colNames);

var row = new Array();
row[0] = '413a4d69-becb-4ae4-8fdd-980755d6a7fb';//normally retrieved via security.getElementUUIDs(...)
row[1] = JSSecurity.VIEWABLE|JSSecurity.ACCESSIBLE; // use bitwise 'or' for both
dataset.addRow(row);//setting element security

row = new Array();
row[0] = 'example_data.orders';
row[1] = JSSecurity.READ|JSSecurity.INSERT|JSSecurity.UPDATE|JSSecurity.DELETE|JSSecurity.TRACKING; //use
bitwise 'or' for multiple flags
dataset.addRow(row);//setting table security

security.setSecuritySettings(dataset);//to be called in solution startup method
```

UPDATE

Constant representing the update flag for table security.

Returns

Number

Sample

```
var colNames = new Array();
colNames[0] = 'uuid';
colNames[1] = 'flags';
var dataset = databaseManager.createEmptyDataSet(0,colNames);

var row = new Array();
row[0] = '413a4d69-becb-4ae4-8fdd-980755d6a7fb';//normally retrieved via security.getElementUUIDs(...)
row[1] = JSSecurity.VIEWABLE|JSSecurity.ACCESSIBLE; // use bitwise 'or' for both
dataset.addRow(row);//setting element security

row = new Array();
row[0] = 'example_data.orders';
row[1] = JSSecurity.READ|JSSecurity.INSERT|JSSecurity.UPDATE|JSSecurity.DELETE|JSSecurity.TRACKING; //use
bitwise 'or' for multiple flags
dataset.addRow(row);//setting table security

security.setSecuritySettings(dataset);//to be called in solution startup method
```

VIEWABLE

Constant representing the viewable flag for form security.

Returns

Number

Sample

```
var colNames = new Array();
colNames[0] = 'uuid';
colNames[1] = 'flags';
var dataset = databaseManager.createEmptyDataSet(0,colNames);

var row = new Array();
row[0] = '413a4d69-becb-4ae4-8fdd-980755d6a7fb';//normally retrieved via security.getElementUUIDs(...)
row[1] = JSSecurity.VIEWABLE|JSSecurity.ACCESSIBLE; // use bitwise 'or' for both
dataset.addRow(row);//setting element security

row = new Array();
row[0] = 'example_data.orders';
row[1] = JSSecurity.READ|JSSecurity.INSERT|JSSecurity.UPDATE|JSSecurity.DELETE|JSSecurity.TRACKING; //use
bitwise 'or' for multiple flags
dataset.addRow(row);//setting table security

security.setSecuritySettings(dataset);//to be called in solution startup method
```

Method Details

addUserToGroup

Boolean **addUserToGroup**(a_userUID, groupName)

Adds an user to a named group.

Parameters

{**Object**} a_userUID – the user UID to be added

{**Object**} groupName – the group to add to

Returns

Boolean – true if added

Sample

```
var userUID = security.getUserUID();
security.addUserToGroup(userUID, 'groupname');
```

authenticate

Object **authenticate**(authenticator_solution, method)

Authenticate to the Servoy Server using one of the installed authenticators or the Servoy default authenticator.

Note: this method should be called from a login solution.

Parameters

{**String**} authenticator_solution – authenticator solution installed on the Servoy Server, null for servoy built-in authentication

{**String**} method – authenticator method, null for servoy built-in authentication

Returns

Object – authentication result from authenticator solution or boolean in case of servoy built-in authentication

Sample

```
// create the credentials object as expected by the authenticator solution
var ok = security.authenticate('myldap_authenticator', 'login', [scopes.globals.userName, scopes.globals.
passWord])
if (!ok)
{
    plugins.dialogs.showErrorDialog('Login failed', 'OK')
}

// if no authenticator name is used, the credentials are checked using the Servoy built-in user management
ok = security.authenticate(null, null, [scopes.globals.userName, scopes.globals.passWord])
```

authenticate

Object **authenticate**(authenticator_solution, method, credentials)

Authenticate to the Servoy Server using one of the installed authenticators or the Servoy default authenticator.

Note: this method should be called from a login solution, once logged in, the authenticate method has no effect.

Parameters

{String} authenticator_solution – authenticator solution installed on the Servoy Server, null for servoy built-in authentication

{String} method – authenticator method, null for servoy built-in authentication

{Object[]} credentials – array whose elements are passed as arguments to the authenticator method, in case of servoy built-in authentication this should be [username, password]

Returns

Object – authentication result from authenticator solution or boolean in case of servoy built-in authentication

Sample

```
// create the credentials object as expected by the authenticator solution
var ok = security.authenticate('myldap_authenticator', 'login', [scopes.globals.userName, scopes.globals.
passWord])
if (!ok)
{
    plugins.dialogs.showErrorDialog('Login failed', 'OK')
}

// if no authenticator name is used, the credentials are checked using the Servoy built-in user management
ok = security.authenticate(null, null, [scopes.globals.userName, scopes.globals.passWord])
```

canDelete

Boolean **canDelete**(dataSource)

Returns a boolean value for security rights.

Parameters

{String} dataSource – the datasource

Returns

Boolean – true if allowed

Sample

```
var dataSource = controller.getDataSource();
var canDelete = security.canDelete(dataSource);
var canInsert = security.canInsert(dataSource);
var canUpdate = security.canUpdate(dataSource);
var canRead = security.canRead(dataSource);
application.output("Can delete? " + canDelete);
application.output("Can insert? " + canInsert);
application.output("Can update? " + canUpdate);
application.output("Can read? " + canRead);
```

canInsert

Boolean **canInsert**(dataSource)

Returns a boolean value for security rights.

Parameters

{String} dataSource – the datasource

Returns

Boolean – true if allowed

Sample

```
var dataSource = controller.getDataSource();
var canDelete = security.canDelete(dataSource);
var canInsert = security.canInsert(dataSource);
var canUpdate = security.canUpdate(dataSource);
var canRead = security.canRead(dataSource);
application.output("Can delete? " + canDelete);
application.output("Can insert? " + canInsert);
application.output("Can update? " + canUpdate);
application.output("Can read? " + canRead);
```

canRead

Boolean **canRead**(dataSource)

Returns a boolean value for security rights.

Parameters

{String} dataSource – the datasource

Returns

Boolean – true if allowed

Sample

```
var dataSource = controller.getDataSource();
var canDelete = security.canDelete(dataSource);
var canInsert = security.canInsert(dataSource);
var canUpdate = security.canUpdate(dataSource);
var canRead = security.canRead(dataSource);
application.output("Can delete? " + canDelete);
application.output("Can insert? " + canInsert);
application.output("Can update? " + canUpdate);
application.output("Can read? " + canRead);
```

canUpdate

Boolean canUpdate(dataSource)

Returns a boolean value for security rights.

Parameters

{String} dataSource – the datasource

Returns

Boolean – true if allowed

Sample

```
var dataSource = controller.getDataSource();
var canDelete = security.canDelete(dataSource);
var canInsert = security.canInsert(dataSource);
var canUpdate = security.canUpdate(dataSource);
var canRead = security.canRead(dataSource);
application.output("Can delete? " + canDelete);
application.output("Can insert? " + canInsert);
application.output("Can update? " + canUpdate);
application.output("Can read? " + canRead);
```

changeGroupName

Boolean changeGroupName(oldGroupName, newGroupName)

Changes the groupname of a group.

Parameters

{Object} oldGroupName – the old name

{String} newGroupName – the new name

Returns

Boolean – true if changed

Sample

```
security.changeGroupName('oldGroup', 'newGroup');
```

changeUserName

Boolean changeUserName(a_userUID, username)

Changes the username of the specified userUID.

Parameters

{Object} a_userUID – the userUID to work on

{String} username – the new username

Returns

Boolean – true if changed

Sample

```
if(security.changeUserName(security.getUserUID('name1'), 'name2'))
{
    application.output('Username changed');
}
```

checkPassword

Boolean checkPassword(a_userUID, password)

Returns true if the password for that userUID is correct, else false.

Parameters

{Object} a_userUID – the userUID to check the password for

{String} password – the new password

Returns

Boolean – true if password oke

Sample

```
if(security.checkPassword(security.getUserUID(), 'password1'))
{
    security.setPassword(security.getUserUID(), 'password2')
}
else
{
    application.output('wrong password')
}
```

createGroup

String createGroup(groupName)

Creates a group, returns the groupname (or null when group couldn't be created).

Parameters

{String} groupName – the group name to create

Returns

String – the created groupname

Sample

```
var deleteGroup = true;
//ceate a group
var groupName = security.creategroup('myGroup');
if (groupName)
{
    //create a user
    var uid = security.createUser('myusername', 'mypassword');
    if (uid) //test if user was created
    {
        //set a newUID for the user
        var isChanged = security.setUserUID(uid, 'myUserUID')
        // add user to group
        security.addUserToGroup(uid, groupName);
        // if not delete group, do delete group
        if (deleteGroup)
        {
            security.deleteGroup(groupName);
        }
    }
}
```

createUser

Object createUser(username, password)

Creates a new user, returns new uid (or null when group couldn't be created or user already exist).

Parameters

{String} username – the username

{String} password – the user password

Returns

Object – the userUID the created userUID, will be same if provided

Sample

```
var removeUser = true;
//create a user
var uid = security.createUser('myusername', 'mypassword');
if (uid) //test if user was created
{
    // Get all the groups
    var set = security.getGroups();
    for(var p = 1 ; p <= set.getMaxRowIndex() ; p++)
    {
        // output name of the group
        application.output(set.getValue(p, 2));
        // add user to group
        security.addUserToGroup(uid, set.getValue(p,2));
    }
    // if not remove user, remove user from all the groups
    if(!removeUser)
    {
        // get now all the groups that that users has (all if above did go well)
        var set =security.getUserGroups(uid);
        for(var p = 1;p<=set.getMaxRowIndex();p++)
        {
            // output name of the group
            application.output(set.getValue(p, 2));
            // remove the user from the group
            security.removeUserFromGroup(uid, set.getValue(p,2));
        }
    }
    else
    {
        // delete the user (the user will be removed from the groups)
        security.deleteUser(uid);
    }
}
```

createUser

Object **createUser**(username, password, userID)

Creates a new user, returns new uid (or null when group couldn't be created or user already exist).

Parameters

{String} username – the username

{String} password – the user password

{Object} userID – the user UID to use

Returns

Object – the userID the created userUID, will be same if provided

Sample

```
var removeUser = true;
//create a user
var uid = security.createUser('myusername', 'mypassword');
if (uid) //test if user was created
{
    // Get all the groups
    var set = security.getGroups();
    for(var p = 1 ; p <= set.getMaxRowIndex() ; p++)
    {
        // output name of the group
        application.output(set.getValue(p, 2));
        // add user to group
        security.addUserToGroup(uid, set.getValue(p,2));
    }
    // if not remove user, remove user from all the groups
    if(!removeUser)
    {
        // get now all the groups that that users has (all if above did go well)
        var set =security.getUserGroups(uid);
        for(var p = 1;p<=set.getMaxRowIndex();p++)
        {
            // output name of the group
            application.output(set.getValue(p, 2));
            // remove the user from the group
            security.removeUserFromGroup(uid, set.getValue(p,2));
        }
    }
    else
    {
        // delete the user (the user will be removed from the groups)
        security.deleteUser(uid);
    }
}
```

deleteGroup

Boolean deleteGroup(groupName)

Deletes a group, returns true if no error was reported.

Parameters

{Object} groupName – the name of the group to delete

Returns

Boolean – true if deleted

Sample

```
var deleteGroup = true;
//ceate a group
var groupName = security.creategroup('myGroup');
if (groupName)
{
    //create a user
    var uid = security.createUser('myusername', 'mypassword');
    if (uid) //test if user was created
    {
        //set a newUID for the user
        var isChanged = security.setUserUID(uid,'myUserUID')
        // add user to group
        security.addUserToGroup(uid, groupName);
        // if not delete group, do delete group
        if (deleteGroup)
        {
            security.deleteGroup(groupName);
        }
    }
}
```

deleteUser

Boolean `deleteUser(userID)`

Deletes an user. returns true if no error was reported.

Parameters

{**Object**} `userID` – The UID of the user to be deleted.

Returns

Boolean – true if the user is successfully deleted.

Sample

```
var removeUser = true;
//create a user
var uid = security.createUser('myusername', 'mypassword');
if (uid) //test if user was created
{
    // Get all the groups
    var set = security.getGroups();
    for(var p = 1 ; p <= set.getMaxRowIndex() ; p++)
    {
        // output name of the group
        application.output(set.getValue(p, 2));
        // add user to group
        security.addUserToGroup(uid, set.getValue(p,2));
    }
    // if not remove user, remove user from all the groups
    if(!removeUser)
    {
        // get now all the groups that that users has (all if above did go well)
        var set =security.getUserGroups(uid);
        for(var p = 1;p<=set.getMaxRowIndex();p++)
        {
            // output name of the group
            application.output(set.getValue(p, 2));
            // remove the user from the group
            security.removeUserFromGroup(uid, set.getValue(p,2));
        }
    }
    else
    {
        // delete the user (the user will be removed from the groups)
        security.deleteUser(uid);
    }
}
```

`getClientID`

String `getClientID()`

Returns the client ID.

Returns

String – the clientId as seen on the server admin page

Sample

```
var clientId = security.getClientID()
```

`getElementUUIDs`

JSDataSet `getElementUUIDs(formname)`

Returns the form elements UUID's as dataset, the one with no name is the form itself.

Parameters

{**String**} `formname` – the formname to retrieve the dataset for

Returns

JSDataSet – dataset with element info

Sample

```
var formElementsUUIDDataSet = security.getElementUUIDs('orders_form');
```

`getGroups`

JSDataSet `getGroups()`

Get all the groups (returns a dataset).
first id column is deprecated!, use only the group name column.

Returns

[JSDataSet](#) – dataset with all the groups

Sample

```
var removeUser = true;
//create a user
var uid = security.createUser('myusername', 'mypassword');
if (uid) //test if user was created
{
    // Get all the groups
    var set = security.getGroups();
    for(var p = 1 ; p <= set.getMaxRowIndex() ; p++)
    {
        // output name of the group
        application.output(set.getValue(p, 2));
        // add user to group
        security.addUserToGroup(uid, set.getValue(p,2));
    }
    // if not remove user, remove user from all the groups
    if(!removeUser)
    {
        // get now all the groups that that users has (all if above did go well)
        var set =security.getUserGroups(uid);
        for(var p = 1;p<=set.getMaxRowIndex();p++)
        {
            // output name of the group
            application.output(set.getValue(p, 2));
            // remove the user from the group
            security.removeUserFromGroup(uid, set.getValue(p,2));
        }
    }
    else
    {
        // delete the user (the user will be removed from the groups)
        security.deleteUser(uid);
    }
}
```

[getSystemUserName](#)

[String](#) **getSystemUserName()**

Retrieves the username of the currently logged in user on operating system level.

Returns

[String](#) – the os user name

Sample

```
//gets the current os username
var osUserName = security.getSystemUserName();
```

[getUserGroups](#)

[JSDataSet](#) **getUserGroups()**

Get all the groups of the current user.

Returns

[JSDataSet](#) – dataset with groupnames

Sample

```
//get all the users in the security settings (Returns a JSDataset)
var dsUsers = security.getUsers()

//loop through each user to get their group
//The getValue call is (row,column) where column 1 == id and 2 == name
for(var i=1 ; i<=dsUsers.getMaxRowIndex() ; i++)
{
    //print to the output debugger tab: "user: " and the username
    application.output("user:" + dsUsers.getValue(i,2));

    //set p to the user group for the current user
    /** @type {JSDataset} */
    var p = security.getUserGroups(dsUsers.getValue(i,1));

    for(k=1;k<=p.getMaxRowIndex();k++)
    {
        //print to the output debugger tab: "group" and the group(s)
        //the user belongs to
        application.output("group: " + p.getValue(k,2));
    }
}
```

getUserGroups

JSDataset **getUserGroups**(userID)

Get all the groups for given user UID.

Parameters

{Object} userID – to retrieve the user groups

Returns

JSDataset – dataset with groupnames

Sample

```
//get all the users in the security settings (Returns a JSDataset)
var dsUsers = security.getUsers()

//loop through each user to get their group
//The getValue call is (row,column) where column 1 == id and 2 == name
for(var i=1 ; i<=dsUsers.getMaxRowIndex() ; i++)
{
    //print to the output debugger tab: "user: " and the username
    application.output("user:" + dsUsers.getValue(i,2));

    //set p to the user group for the current user
    /** @type {JSDataset} */
    var p = security.getUserGroups(dsUsers.getValue(i,1));

    for(k=1;k<=p.getMaxRowIndex();k++)
    {
        //print to the output debugger tab: "group" and the group(s)
        //the user belongs to
        application.output("group: " + p.getValue(k,2));
    }
}
```

getUserName

String **getUserName**()

Get the current user name (null if not logged in), finds the user name for given user UID if passed as parameter.

Returns

String – the user name

Sample

```
//gets the current loggedIn username
var userName = security.getUserName();
```

getUserName

String **getUserName**(userID)

Get the current user name (null if not logged in), finds the user name for given user UID if passed as parameter.

Parameters

{Object} userID – the user UID used to retrieve the name

Returns

String – the user name

Sample

```
//gets the current loggedIn username
var userName = security.getUserName();
```

getUserUID

String **getUserUID**()

Get the current user UID (null if not logged in); finds the userID for given user_name if passed as parameter.

Returns

String – the userID

Sample

```
//gets the current loggedIn username
var userName = security.getUserName();
//gets the uid of the given username
var userID = security.getUserUID(userName);
//is the same as above
//var my_userUID = security.getUserUID();
```

getUserUID

String **getUserUID**(username)

Get the current user UID (null if not logged in); finds the userID for given user_name if passed as parameter.

Parameters

{String} username – the username to find the userID for

Returns

String – the userID

Sample

```
//gets the current loggedIn username
var userName = security.getUserName();
//gets the uid of the given username
var userID = security.getUserUID(userName);
//is the same as above
//var my_userUID = security.getUserUID();
```

getUsers

JSDataset **getUsers**()

Get all the users in the security settings (returns a dataset).

Returns

JSDataset – dataset with all the users

Sample

```
//get all the users in the security settings (Returns a JSDataset)
var dsUsers = security.getUsers()

//loop through each user to get their group
//The getValue call is (row,column) where column 1 == id and 2 == name
for(var i=1 ; i<=dsUsers.getMaxRowIndex() ; i++)
{
    //print to the output debugger tab: "user: " and the username
    application.output("user:" + dsUsers.getValue(i,2));

    //set p to the user group for the current user
    /** @type {JSDataset} */
    var p = security.getUserGroups(dsUsers.getValue(i,1));

    for(k=1;k<=p.getMaxRowIndex();k++)
    {
        //print to the output debugger tab: "group" and the group(s)
        //the user belongs to
        application.output("group: " + p.getValue(k,2));
    }
}
```

getUsers

JSDataset **getUsers**(groupName)

Get all the users in the security settings (returns a dataset).

Parameters

{String} groupName – the group to filter on

Returns

JSDataset – dataset with all the users

Sample

```
//get all the users in the security settings (Returns a JSDataset)
var dsUsers = security.getUsers()

//loop through each user to get their group
//The getValue call is (row,column) where column 1 == id and 2 == name
for(var i=1 ; i<=dsUsers.getMaxRowIndex() ; i++)
{
    //print to the output debugger tab: "user: " and the username
    application.output("user:" + dsUsers.getValue(i,2));

    //set p to the user group for the current user
    /** @type {JSDataset} */
    var p = security.getUserGroups(dsUsers.getValue(i,1));

    for(k=1;k<=p.getMaxRowIndex();k++)
    {
        //print to the output debugger tab: "group" and the group(s)
        //the user belongs to
        application.output("group: " + p.getValue(k,2));
    }
}
```

isUserMemberOfGroup

Boolean **isUserMemberOfGroup**(groupName)

Check whatever the current user is part of the specified group

Parameters

{String} groupName – name of the group to check

Returns

Boolean – dataset with groupnames

Sample

```
//check whatever user is part of the Administrators group
if(security.isUserMemberOfGroup('Administrators', security.getUserUID('admin'))){
    // do administration stuff
}
```

isUserMemberOfGroup

Boolean **isUserMemberOfGroup**(groupName, userID)

Check whatever the user specified as parameter is part of the specified group.

Parameters

{String} groupName – name of the group to check

{Object} userID – UID of the user to check

Returns

Boolean – dataset with groupnames

Sample

```
//check whatever user is part of the Administrators group
if(security.isUserMemberOfGroup('Administrators', security.getUserUID('admin'))){
    // do administration stuff
}
```

login

Boolean **login**(username, a_userUID, groups)

Login to be able to leave the solution loginForm.

Example: Group names may be received from LDAP (Lightweight Directory Access Protocol) - a standard protocol used in web browsers and email applications to enable lookup queries that access a directory listing.

Parameters

{String} username – the username, like 'JamesWebb'

{Object} a_userUID – the user UID to process login for

{String[]} groups – the groups array

Returns

Boolean – true if loggedin

Sample

```
var groups = ['Administrators']; //normally these groups are for example received from LDAP
var user_uid = scopes.globals.email; //also this uid might be received from external authentication method
var ok = security.login(scopes.globals.username, user_uid , groups)
if (!ok)
{
    plugins.dialogs.showErrorDialog('Login failure', 'Already logged in? or no user_uid/groups specified?', 'OK')
}
```

logout

void **logout**()

Logout the current user and close the solution, if the solution requires authentication and user is logged in.

You can redirect to another solution if needed; if you want to go to a different url, you need to call application.showURL(url) before calling security.logout() (this is only applicable for Web Client).

An alternative option to close a solution and to open another solution, while keeping the user logged in, is application.closeSolution().

Returns

void

Sample

```
//Set the url to go to after logout.
//application.showURL('http://www.servoy.com', '_self'); //Web Client only
security.logout();
//security.logout('solution_name');//log out and close current solution and open solution 'solution_name'
//security.logout('solution_name','global_method_name','my_argument');//log out, close current solution, open
solution 'solution_name', call global method 'global_method_name' with argument 'my_argument'
//Note: specifying a solution will not work in the Developer due to debugger dependencies
//specified solution should be of compatible type with client (normal type or client specific(Smart client only
/Web client only) type )
```

logout

void **logout**(solutionToLoad)

Logout the current user and close the solution, if the solution requires authentication and user is logged in.

You can redirect to another solution if needed; if you want to go to a different url, you need to call application.showURL(url) before calling security.logout() (this is only applicable for Web Client).

An alternative option to close a solution and to open another solution, while keeping the user logged in, is application.closeSolution().

Parameters

{String} solutionToLoad – the solution to load after logout

Returns

void

Sample

```
//Set the url to go to after logout.
//application.showURL('http://www.servoy.com', '_self'); //Web Client only
security.logout();
//security.logout('solution_name');//log out and close current solution and open solution 'solution_name'
//security.logout('solution_name','global_method_name','my_argument');//log out, close current solution, open
solution 'solution_name', call global method 'global_method_name' with argument 'my_argument'
//Note: specifying a solution will not work in the Developer due to debugger dependencies
//specified solution should be of compatible type with client (normal type or client specific(Smart client only
/Web client only) type )
```

logout

void **logout**(solutionToLoad, method)

Logout the current user and close the solution, if the solution requires authentication and user is logged in.

You can redirect to another solution if needed; if you want to go to a different url, you need to call application.showURL(url) before calling security.logout() (this is only applicable for Web Client).

An alternative option to close a solution and to open another solution, while keeping the user logged in, is application.closeSolution().

Parameters

{String} solutionToLoad – the solution to load after logout

{String} method – the method to run in the solution to load

Returns

void

Sample

```
//Set the url to go to after logout.
//application.showURL('http://www.servoy.com', '_self'); //Web Client only
security.logout();
//security.logout('solution_name');//log out and close current solution and open solution 'solution_name'
//security.logout('solution_name','global_method_name','my_argument');//log out, close current solution, open
solution 'solution_name', call global method 'global_method_name' with argument 'my_argument'
//Note: specifying a solution will not work in the Developer due to debugger dependencies
//specified solution should be of compatible type with client (normal type or client specific(Smart client only
/Web client only) type )
```

logout

void **logout**(solutionToLoad, method, argument)

Logout the current user and close the solution, if the solution requires authentication and user is logged in.

You can redirect to another solution if needed; if you want to go to a different url, you need to call application.showURL(url) before calling security.logout() (this is only applicable for Web Client).

An alternative option to close a solution and to open another solution, while keeping the user logged in, is application.closeSolution().

Parameters

[{String}](#) solutionToLoad – the solution to load after logout
[{String}](#) method – the method to run in the solution to load
[{Object}](#) argument – the argument to pass to the method to run

Returns

void

Sample

```
//Set the url to go to after logout.
//application.showURL('http://www.servoy.com', '_self'); //Web Client only
security.logout();
//security.logout('solution_name');//log out and close current solution and open solution 'solution_name'
//security.logout('solution_name','global_method_name','my_argument');//log out, close current solution, open
solution 'solution_name', call global method 'global_method_name' with argument 'my_argument'
//Note: specifying a solution will not work in the Developer due to debugger dependencies
//specified solution should be of compatible type with client (normal type or client specific(Smart client only
/Web client only) type )
```

removeUserFromGroup

[Boolean](#) **removeUserFromGroup**(a_userUID, groupName)

Removes an user from a group.

Parameters

[{Object}](#) a_userUID – the user UID to be removed
[{Object}](#) groupName – the group to remove from

Returns

[Boolean](#) – true if removed

Sample

```
var removeUser = true;
//create a user
var uid = security.createUser('myusername', 'mypassword');
if (uid) //test if user was created
{
    // Get all the groups
    var set = security.getGroups();
    for(var p = 1 ; p <= set.getMaxRowIndex() ; p++)
    {
        // output name of the group
        application.output(set.getValue(p, 2));
        // add user to group
        security.addUserToGroup(uid, set.getValue(p,2));
    }
    // if not remove user, remove user from all the groups
    if(!removeUser)
    {
        // get now all the groups that that users has (all if above did go well)
        var set =security.getUserGroups(uid);
        for(var p = 1;p<=set.getMaxRowIndex();p++)
        {
            // output name of the group
            application.output(set.getValue(p, 2));
            // remove the user from the group
            security.removeUserFromGroup(uid, set.getValue(p,2));
        }
    }
    else
    {
        // delete the user (the user will be removed from the groups)
        security.deleteUser(uid);
    }
}
```

setPassword

[Boolean](#) **setPassword**(a_userUID, password)

Set a new password for the given userUID.

Parameters

[{Object}](#) a_userUID – the userUID to set the new password for
[{String}](#) password – the new password

Returns

Boolean – true if changed

Sample

```
if (security.checkPassword(security.getUserUID(), 'password1'))
{
    security.setPassword(security.getUserUID(), 'password2')
}
else
{
    application.output('wrong password')
}
```

setSecuritySettings

void **setSecuritySettings**(dataset)

Sets the security settings; the entries contained in the given dataset will override those contained in the current security settings.

NOTE: The security.getElementUIDs and security.setSecuritySettings functions can be used to define custom security that overrides Servoy security. For additional information see the function security.getElementUIDs.

Parameters

{**Object**} dataset – the dataset with security settings

Returns

void

Sample

```
var colNames = new Array();
colNames[0] = 'uuid';
colNames[1] = 'flags';
var dataset = databaseManager.createEmptyDataSet(0,colNames);

var row = new Array();
row[0] = '413a4d69-becb-4ae4-8fdd-980755d6a7fb'; //normally retrieved via security.getElementUIDs(...)
row[1] = JSSecurity.VIEWABLE|JSSecurity.ACCESSIBLE; // use bitwise 'or' for both
dataset.addRow(row); //setting element security

row = new Array();
row[0] = 'example_data.orders';
row[1] = JSSecurity.READ|JSSecurity.INSERT|JSSecurity.UPDATE|JSSecurity.DELETE|JSSecurity.TRACKING; //use
bitwise 'or' for multiple flags
dataset.addRow(row); //setting table security

security.setSecuritySettings(dataset); //to be called in solution startup method
```

setUserID

Boolean **setUserID**(a_userUID, newUserUID)

Set a new userID for the given userID.

Parameters

{**Object**} a_userUID – the userID to set the new user UID for

{**String**} newUserUID – the new user UID

Returns

Boolean – true if changed

Sample

```
var deleteGroup = true;
//ceate a group
var groupName = security.creategroup('myGroup');
if (groupName)
{
    //create a user
    var uid = security.createUser('myusername', 'mypassword');
    if (uid) //test if user was created
    {
        //set a newUID for the user
        var isChanged = security.setUserUID(uid, 'myUserUID')
        // add user to group
        security.addUserToGroup(uid, groupName);
        // if not delete group, do delete group
        if (deleteGroup)
        {
            security.deleteGroup(groupName);
        }
    }
}
```