# WS Functional Automation

# 8.15 WSODBC

## Purpose

With WSODBC, we can make ODBC connections with Liquid UI WS. This article explains us how to use the ODBC functionality built into Liquid UI WS.

You can use the Liquid UI WS platform to access data and manipulate functions in Microsoft SQL Server and other ODBC-compliant databases through the wsodbc extension. WSODBC is a set of libraries that you can use to make ODBC connections and perform SQL queries from within Liquid UI WS scripts. Currently, this functionality is in use in the field by several of Synactive customers, most notably Florida Crystals.

The WSODBC functionality is available for use with the following Synactive solutions:

- Server-based Implementation (Liquid UI Server)
- Web-based Implementation (Web Server)
- Local Implementation (Liquid UI for SAP GUI)

Through Liquid UI Server, the ODBC functionality can be used with Synactive's Offline and Mobile solutions in addition to the local and Server-based implementations already mentioned. This highlights one of the great strengths of WS - you can write a single script and implement it with multiple touchpoints. With WSODBC, you can use the exec() method of your database object to execute any SQL commands that are supported by Microsoft SQL Server. Some of the most common commands, which we will be demonstrating in our examples, are the following.

**Specify Databases**
: You can specify more than one database to connect to. However, you can only connect to a single database at a time - simultaneous connections are not currently possible.

**Create Tables**
: You can use wsodbc to create new tables in SQL databases. In addition, you can add rows to either the newly created tables or existing tables in the database.

**Retrieve Data**
: You can use SQL 'Select' statements to retrieve data from within SQL databases. The data will be returned as a relational array, so once returned, you can then perform operations upon it from with the WS script.

**Update Databases**
: You can insert data into database tables with the wsodbc extension. In addition, you can specify individual table within databases. You can also add entries to database tables with wsodbc.

**Delete Table Entries**
: You can use wsodbc to delete individual entries from a database table.

**Drop Tables**
: You can use wsodbc to drop an entire table from a given database.

# WS Functional Automation

The ODBC functionality is accomplished with an extension to the WS platform called wsodbc. In this section, we will explain how to use the GuiXT WS to work with SQL databases. The following topics will be covered:



### ODBC Requirements

The system requirements for running ODBC with Liquid UI WS is as follows:

The GuiXT WS engine must be installed in order to utilize the ODBC functionality. In addition, you also need to install the following file:

- **wsodbc.dll**: Contains the libraries for the SQL connection.

To install the wsodbc.dll, copy it to the correct directory. Please note that the above file must be stored in the same folder as the GuiXT WS engine. These directories are as follows.

### Local Deployment (GuiXT WS Desktop or Liquid UI for SAP GUI)

For local GuiXT WS installations, please install the wsodbc.dll file to the C:\Program Files\SAP\FrontEnd\SAPgui\ directory.

### Server Deployments

For deployments involving GuiXT Server, please place the wsodbc.dll file in the C:\Program Files\Synactive Inc\GuiXTServer\ directory.

### Web Deployment

For deployments using Web Server, the wsodbc.dll file should go in the C:\Program Files\Synactive Inc\GuiXTFuzion\ directory.

### Connection Parameters

You must specify connection parameters before you can create a connection object for wsodbc. Please follow the instructions below to set up connection parameters for a connection object.

1. Load the WSODBC libraries:

   ```
   load('wsodbc.dll');
   ```

2. Create the object for the connection parameters:

   ```
   var dbtest ={Driver={SQLNativeClient};server:'NAME',dbname:'TEST
   ,user:'test',pass:'test'};
   ```

   These parameters are defined as follows:

   - **Driver:** The driver you will use. You must have the appropriate driver for the version of the SQL server and for the database you will be connecting to. Check your server and ensure that you have the appropriate native SQL driver installed on the machine that will be connecting to the SQL server. You can find additional information about the drivers from Microsoft's MSDN Library.
   - **server:** The name of the database server to which you will be connecting. This can be either the name of the server or the server's IP address.
   - **dbname:** The name of the specific database to which you will be connecting.
   - **user:** The name of the specific user who will be connecting to the SQL server.
   - **pass:** The password of the specific user who will be connecting to the SQL server.

   **Note**: The connection parameters also can be passed to the connection object from a variable.

3. Proceed to the Creating Connection Objects section.

### Creating Connection Objects

This section describes how to create a connection object.

1. Load the wsodbc.dll as previously described if you have not already done it.
2. Create the connection parameters as previously described.
3. Create the connection object. This will establish a new ODBC connection using the connection parameters you previously defined. An example is shown below:

   ```
   db = new Odbc(dbTest);
   ```

We recommend placing the connection object in a try-catch block so as to ensure proper error handling. An example script is shown below.

```
function ODBCconnect(dbase)
{
  var sConnectTrusted = 'Driver={SQL Native Client};Server={'+db
ase.server+'};Database={'+dbase.dbname+'};'Trusted_Connection:Ye
s'
  var sConnectUser = 'Driver={SQL Native Client'};Server={'+dbas
e.server+'};Database={'+dbase.dbname+'};UID={+dbase.user+'};PWD=
{'+dbase.pass+'}';
  var sConnect = '';
  if(dbase.user) sConnect = sConnectUser;
  else sCOnnect = sConnectTrusted;
  println('Connecting with '+sConnect);
  try{
    db = new Odbc(sConnect);
    }
    catch(err){
      message("E: Error with database conectivity.");
      println("> error with DB:",err.description);
      return NULL;
    }
  println("
Connected to "+db.dbms+'. Server '+dbase.server+' Database '+dba
se.dbname);
  return db;
}
```

4. Follow the next section, Using SQL create statements:

### Using SQL Create Statements

You can use the SQL CREATE statement in conjunction with wsodbc to perform updates and additions to the database. To do this, you will typically use the db.exec() function, with the SQL statement as an argument. An example of this is shown below:

```
var arrObj = db.exec( "SELECT * FROM TABLE" );
```

**Note**: Only the SELECT statement will return values. The INSERT, DELETE, UPDATE, and CREATE statements do not return any values. If you are returning values from the database, we recommend that you use an array to hold them.

In the following example, we will use the CREATE statement to create a new table in a SQL database.

1. Create a new database with the following code:

   ```
   db = new Obdc(<connection parameters>);
   ```

2. Create a new SQL query in a variable. An example is shown below:

   ```
   sqlQuery = "if not exists ( select * from sysobjects where name=
   'user' and xtype='U')" +
   "CREATE TABLE[dbo].[USER](" + "[RECID][varchar](15)NOT NULL," +
   "[USERNAME][varchar](12)NOT NULL," +
   "[PASSWORD][varchar](40)NOT NULL," + "[CLIENT][varchar](3)NOT NU
   LL," + "[LANG][varchar](2)NULL," +
   "[CREATE_DATA][date]NOT NULL," + "[MODIFIED_DATE][date]NULL," +
   "[CREATED_BY][varchar](10)NOT NULL," +
   "[MODIFIED_BY][varchar](10)NULL," + "[LOCKED][varchar](1)NULL,"
   + "[FIRST_LOGIN][varchar](1)NULL," +
   "[PLANT][varchar](4)NOT NULL," + ")
   ```

   Synactive recommends always using the if not exists(select * from sysobjects where name='user' and xtype='U'); statement when creating tables. This will verify that the table 'user' exists in the specified database. If the 'user' table does not exist, it will be created. We also recommend handling the record IDs, the session information, modification information and messages for each table.

3. Use the db.exec() function to execute the SQL query as shown below:

   ```
   db.exec(sqlQuery);
   ```

4. To add an entry to the table, create a new query to use in the db.exec() function. An example is shown below:

   ```
   sqlQuery = "if not exists(select * from [dbo].[USER] where usern
   ame = 'admin')
   insert [dbo].[USER] values ('"+ system.newRecId('$S1')+ "','admi
   n','adminpass','100','EN','10/12/2012','',
   'system','','','X','1000','1000')"
   ```

5. Use the db.exec() function to execute the SQL query as shown below:

   ```
   db.exec(sqlQuery);
   ```

   The 'if not exists' statement will verify that the user 'admin' exists. If the user does not exist, the new entry will be added to the table.

```
sqlQuery = "if not exists (select * from [dbo].[USER] WHERE USER
NAME = 'admin') insert [dbo].[USER] VALUES (?,?,?,?,?,?,?,?,?,?,
?,?,?)"
          db.exec(sqlQuery,arrSQLQUERY);
```

6. Because the values are hard-coded in the example above, there is a possibility that the RecID may return a value with a quote. This would break the sequence and generate an incorrect result. To avoid this possibility, you should use an array as shown below:

```
var arrSQLQUERY = [];
arrSQLQUERY = [system.newRecId('$S1'),'admin','adminpass','100',
'EN','10/12/2012','',
'system','','','X','1000','1000'];
```

7. Your SQL query will appear as shown below:

```
sqlQuery = "if not exists (select * from [dbo].[USER] WHERE USER
NAME = 'admin') insert [dbo].[USER] VALUES (?,?,?,?,?,?,?,?,?,?,
?,?,?)"
```

8. Use the db.exec() function to execute your SQL query as shown below:

```
db.exec(sqlQuery,arrSQLQUERY);
```

### Deleting a Connection

To delete a previously created database connection, please do the following.

1. Add the following code to your script.

```
db = NULL
```

2. The connection contained in the variable 'db' is now destroyed.

### Retrieving Data

To retrieve data from a database, you will use the SQL SELECT statement in your sql query. As this statement will return values, you must have code to handle the values returned. To perform a retrieve operation, you can use the db.exec() function that we previously introduced, or you can use the db.select() function. Both will return data when used with a SELECT statement. We will demonstrate this in the following examples.

1. To use a db.exec() statement, create the SQL query as a variable as in the following example:

```
sqlQuery = "select * from [db0].[USER]";
```

2. Create the variable as shown below:

```
var record = db.select(sqlQuery);
```

3. Execute the query and then access the values returned. In this example, we will

```
var vUser = record[0].USER
var vClient = record[0].CLIENT
```

**Note**: The values returned from a database by a SELECT statement will be in the form of a dimensional array. You can include loops to get each column and row element in the table. We also recommend using try-catch blocks to handle any errors that may occur.

### Updating Database Tables

To retrieve data from a database, you will use the SQL SELECT statement in your sql query. As this statement will return values, you must have code to handle the values returned. To perform a retrieve operation, you can use the db.exec() function that we previously introduced, or you can use the db.select() function. Both will return data when used with a SELECT statement. We will demonstrate this in the following examples.

1. To use a db.exec() statement, create the SQL query as a variable as in the following example:

```
sqlQuery = "select * from [db0].[USER]";
```

2. Create the variable as shown below:

```
var record = db.select(sqlQuery);
```

3. Execute the query and then access the values returned. In this example, we will

```
var vUser = record[0].USER
var vClient = record[0].CLIENT
```

**Note**: The values returned from a database by a SELECT statement will be in the form of a dimensional array. You can include loops to get each column and row element in the table. We also recommend using try-catch blocks to handle any errors that may occur.

### Deleting a Record

To retrieve data from a database, you will use the SQL SELECT statement in your sql query. As this statement will return values, you must have code to handle the values returned. To perform a retrieve operation, you can use the db.exec() function that we previously introduced, or you can use the db.select() function. Both will return data when used with a SELECT statement. We will demonstrate this in the following examples.

1. To use a db.exec() statement, create the SQL query as a variable as in the following example:

   ```
   sqlQuery = "select * from [db0].[USER]";
   ```

2. Create the variable as shown below:

   ```
   var record = db.select(sqlQuery);
   ```

3. Execute the query and then access the values returned. In this example, we will

   ```
   var vUser = record[0].USER
   var vClient = record[0].CLIENT
   ```

   **Note**: The values returned from a database by a SELECT statement will be in the form of a dimensional array. You can include loops to get each column and row element in the table. We also recommend using try-catch blocks to handle any errors that may occur.

### WSODBC Example

The following examples demostrate the usage of WSODBC in a WS script.

### dbo.sjs

This file contains the global variables and the main functions of the ODBC operation. The included functions are as follows.

- ODBCconnect(dbase)
- opendb()
- closedb()
- userTable(dbase)

- dbTableCreate()

**Example: dbo.sjs**

Explanation of the dbo.sjs file in the WSODBC example.

The dbo.sjs file contains the global variables and the main functions of the ODBC operation. THe connection parameters are also included as they are global as well. The included sections are as follows:

- Connection Parameters
- ODBCconnect(dbase)
- opendb()
- closedb()
- userTable(dbase)

### Connection Parameters

The first section of the dbo.sjs file contains the connection parameters for the ODBC operation. Any global variables used for the operation also can be stored in this section. The connection parameters are as follows.

```
var db1 ={server:'TEST1',dbname:'NOTIFICATION',user:'user1',pass
:'password'};
var db2 ={server:'TEST2',dbname:'WORKORDER',user:'user2',pass:'p
assword1'};
var db3 ={server:'TEST3',dbname:'USER',user:'user3',pass:'passwo
rd2'};
```

Below the variable for determining the database resides an IF statement. This creates the 'ODBCCon' object that holds the ODBC connection parameters and it also performs the database open and table create operations. The code is as follows.

```
if(LOAD_ONCE){
if(!ODBCCon)
ODBCCon = ODBCconnect(dbtest);
if(ODBCCon)
 dbTableCreate();
else
 println("Failed DB Table Create Block");
LOAD_ONCE--;
}
var SESSIONID;
```

### ODBCconnect(dbase)

The ODBCconnect function passes the database name as a parameter and then it calls the database open function. This function also will query to ensure that the requested database actually exists. Finally, it includes code to handle any errors that might occur during these operations. The code is shown below.

```
  function ODBCconnect(dbase)
{
  var dbtest ={server:'TEST2',dbname:'WORKORDER',user:'user2',pa
ss:'password1'};

  try{
  db = new Odbc(dbtest);
 }
 catch(err){
  message("E: " +err.description);
 }
}
```

### opendb()

This function is the one that actually opens a connection to whatever database you specify in the connection parameters. This function is called by the ODBCConnect function introduced above. The code is as follows.

```
function opendb() {
 println(">>1 that.ODBCCon",that.ODBCCon);
    if(!that.ODBCCon) {
  println('Opening session database...');
        that.ODBCCon = ODBCconnect( dbConnectTo );
  println(">>2 DB connected to",that.ODBCCon);
 }
 println(">>3 that.ODBCCon",that.ODBCCon);
 return that.ODBCCon;
}
```

### closedb()

The closedb function is the counterpart to the opendb function introduced

above. This function will close the connection to the database once all operations are complete. The code is shown below.

```
function closedb() {
    /* \/ \/ \/ \/ \/ \/ \/ */
    if(that.ODBCCon) {
 that.ODBCCon = null;
 delete that.ODBCCon;
 }

    // this.name is set by newSession
    print('Connection closed\n');
 enter('/nex')
}
```

**userTable(dbase)**

The userTable function performs the following operations on the database:

- Checks to ensure that the user table exists.
- If the user table does not exist, the function will create the user table.
- If the user table exists, the function will query for the user 'admin'.
- If the user table does not exist, then there can be no 'admin' user, so the function will create the table and then will insert the 'admin' user.

The code is shown below.

```
function userTable(dbase){
 sqlQuery = "if not exists ( select * from sysobjects where name
='user' and xtype='U')" +
"CREATE TABLE[dbo].[USER](" + "[RECID][varchar](15)NOT NULL," +
"[USERNAME][varchar](12)NOT NULL," +
"[PASSWORD][varchar](40)NOT NULL," + "[CLIENT][varchar](3)NOT NU
LL," + "[LANG][varchar](2)NULL," +
"[CREATE_DATA][date]NOT NULL," + "[MODIFIED_DATE][date]NULL," +
"[CREATED_BY][varchar](10)NOT NULL," +
"[MODIFIED_BY][varchar](10)NULL," + "[LOCKED][varchar](1)NULL,"
+ "[FIRST_LOGIN][varchar](1)NULL," +
"[PLANT][varchar](4)NOT NULL," + "
 dbase.exec(sqlQuery);
sqlQuery = "if not exists(select * from [dbo].[USER] where usern
ame = 'admin')
insert [dbo].[USER] values ('"+ system.newRecId('$S1')+ "','admi
```

```
n','adminpass','100','EN','10/12/2012','',
'system','','','X','1000','1000')"
}
```

**FUNCTIONS_ELOGON.sjs**

This script file contains the functional scripts for the ODBC operation. These functions will be called from the main script. The functions contained within are, listed below.

- set_device(param)
- checkUser()
- changeLayout()
- addUser()
- changePassword()
- searchUser()
- modifyUser()
- lockUserToggle(param)
- resetPassword()

The FUNCTIONS_ELOGON.sjs file contains the the operational functions of the ODBC operation. The connection parameters are also included as they are global as well. The included sections are as follows:



**checkUser()**

This function gets all users from the master and then imports them into the XLOGON table. The code is shown below.

```
function checkUser(){
 db = that.ODBCCon;
 if(!isBlank(USERNAME)){
  //Check if the user exists
  sqlQuery = "SELECT * from [dbo].["+objUserTable.name+"] WHERE "+objU
```

```
serTable.cols.user+" = '"+ USERNAME +"'";
  println("------------>sqlQuery:",sqlQuery);
  record = db.select(sqlQuery);
  println("------------> ",record);
  if (record.length == 0){
    return("E: Username does not exist");
  } else if(record.length == 1){
   //if(record[0][objUserTable.cols.client] != CLIENT)
   // return("Client does not match");
   //if(record[0][objUserTable.cols.user] != USERNAME)
   // return("E: Username does not exist");
   if(record[0][objUserTable.cols.password] != PASSWORD)
    return("E: Password does not match");
   if(record[0][objUserTable.cols.lock] == 'X')
    return("E: User is locked");
   if(record[0][objUserTable.cols.firstlogin] == 'X'){
    changeLayout("NEW_PASSWORD");
    return("Enter New password");
   }
   else{
    if(record[0][objUserTable.cols.user]=='admin'){
     changeLayout("ADMIN_PANEL");
    } else if (record[0][objUserTable.cols.manager]=='X'){
     changeLayout("MANAGER_PANEL");
     WHSE = record[0][objUserTable.cols.warehouse];
    } else {
     changeLayout("PLATFORM");
     PLANT = record[0][objUserTable.cols.plant];
     WHSE = record[0][objUserTable.cols.warehouse];
   }
   }

   PASSWORD=Crypto.Encrypt(PASSWORD);
   println("---> " + USER + " authenticated.. Logging in");
   SESSIONID = (new session()).id;
   println(">",SESSIONID);
   sqlQuery = "insert [dbo].["+objSessionTable.name+"] VALUES ('"+USER
NAME+"','"+SESSIONID+"')";
   println(sqlQuery);
   db.exec(sqlQuery);
  }
  else{
   return("E: User does not exist");
  }
 }
 else
  message("E: Fill in all required entry fields");
}
```

**addUser()**

This function adds a user to a specified database. It wil also return error mesages if the requirement are not met, such as the password being an incorrect length. The code is as follows.

```
function addUser(){
 if(isBlank(zuser) || isBlank(zpass) || isBlank(zpassconfirm)) {
  println("---> Error :");
  return("E: Fill in all required details");
 }
 if(isBlank(zclient) || isBlank(zplant) || isBlank(zwhse)){
  println("---> Error :");
  return("E: Fill in all required details");
 }

 println("--->",zpass);
 println("--->",zpassconfirm);

 if(zpass.toString().trim()!=zpassconfirm.toString().trim()){
  zpass ='';
  zpassconfirm = '';
  return("E: Password does not match");
 }

 if(zpass.length < 6) return("E: Password is not long enough (Minimum
length: 6 characters)");
 db = that.ODBCCon;
 record = db.select("select * from [dbo].["+objUserTable.name+"] WHERE
 "+objUserTable.cols.user+" = '"+zuser+"' AND "+objUserTable.cols.clie
nt+"='"+zclient+"'");
 /*if (record.length == 0)
  return("E: Username does not exists");
 else */
 if ( record.length > 1 )
  return("
E: Error with the database! Multiple user for the same client.");
 sqlQuery = "insert [dbo].["+objUserTable.name+"] VALUES ('"+ system.n
ewRecId('$S1')+ "','"+ zuser + "','"+zpass +"','"+ zclient+"','EN',CON
VERT (datetime, GETDATE()),'','"+USERNAME+"','','','X','"+ zplant+"','
"+ zwhse+"','')";
 t = db.exec(sqlQuery);
 println("------> t: ",t);
 //set("V[z*]","");
 changeLayout("ADMIN_PANEL");
 return("S: User "+zuser+" successfully created in Client "+zclient);
```

```
}
```

**changePassword()**

This function will change a user's password. This incorporates the ability to encrypt the password, using our encrypt method. The code is shown below.

```
function changePassword(){
 if(newPass.toString().trim()!=confirmPass.toString().trim()){
  newPass ='';
  confirmPass = '';
  return("E: Password does not match");
 }
 newPass=newPass.toString().trim();
 if(newPass.length < 6) return("
E: Password is not long enough (Minimum length: 6 characters)");
 db = that.ODBCCon;
 var sqlQuery = "UPDATE [dbo].["+objUserTable.name+"] SET ["+objUserTa
ble.cols.rec+"]='"+system.newRecId('$S1')+"',["+objUserTable.cols.pass
word+"] = '"+Crypto.Encrypt(newPass)+"', ["+objUserTable.cols.firstlog
in+"] = '', ["+objUserTable.cols.modifiedOn+"]=CONVERT (datetime, GETD
ATE()),["+objUserTable.cols.modifiedBy+"]='"+USERNAME+"' WHERE "+objUs
erTable.cols.user+" = '" + USERNAME+"'";
 println(sqlQuery);
 t = db.exec(sqlQuery);
 println("------> t: ",t);
 PASSWORD = Crypto.Encrypt(newPass);
 set("V[z*]","");
 changeLayout("PLATFORM");
 return("S: Password changed successfully");
}
```

**searchUser()**

This function will search for a given user in the database and will return a message if the user cannot be found. The code is in the following example.

```
function searchUser(param){
 if(isBlank(zclient.toString().trim()) || isBlank(zuser.toString().tri
m()))
  return("E: Fill in all the required fields");
```

```
 db = that.ODBCCon;
 record = db.select("select * from [dbo].["+objUserTable.name+"] WHERE
 "+objUserTable.cols.user+" = '"+zuser+"' AND "+objUserTable.cols.clie
nt+" = '"+zclient+"'");
 if(record.length == 1){
  zplant = record[0][objUserTable.cols.plant];
  zwhse = record[0][objUserTable.cols.warehouse];
  zlocked = record[0][objUserTable.cols.lock];
  changeLayout(param.screen);
 }
 else if (record.length == 0)
  return("E: Username does not exists");
 else if ( record.length > 1 )
  return("
E: Error with the database! Multiple user for the same client.");
 else
  return("E: Error code 0x01");
}
```

**modifyUser()**

This function will modify an entry for a user, in case you wish to change information such as the plant where he or she is affiliated. The code is shown below.

```
function modifyUser(){
 if(isBlank(zplant.toString().trim()) || isBlank(zwhse.toString().trim
()))
  return("E: Fill in all the required fields");

 db = that.ODBCCon;
 var sqlQuery = "update [dbo].["+objUserTable.name+"] SET ["+objUserTa
ble.cols.plant+"] = '"+zplant+"', ["+objUserTable.cols.warehouse+"] ='
"+zwhse+"' WHERE "+objUserTable.cols.user+" = '" + zuser+"' AND "+objU
serTable.cols.client+" = '"+zclient+"'";
 t = db.exec(sqlQuery);
 println("------> t: ",t);
 set("V[z*]","");
 changeLayout("MODIFY_USER");
 return("S: User data saved successfully");
}
```

**lockUserToggle(param)**

This function will prevent a specified user from logging into the database. The code is as follows.

```
function lockUserToggle(param){
 db = that.ODBCCon;
 var sqlQuery = "update [dbo].["+objUserTable.name+"] SET ["+objUserTa
ble.cols.lock+"] = '"+param.mode+"' WHERE "+objUserTable.cols.user+" =
 '" + zuser+"' AND "+objUserTable.cols.client+" = '"+zclient+"'";
 t = db.exec(sqlQuery);
 if(zlocked == "X"){
  zlocked= " ";
  return("S: User has been unlocked");
 }
 else{
  zlocked= "X";
  return("S: User has been locked");
 }
}
```

**resetPassword()**

This function resets a user's password if he or she forgets it. Error handling is also included in case the reset operation does not succeed. The code is shown below.

```
function resetPassword(){
 if(newPass.toString().trim()!=confirmPass.toString().trim()){
  newPass ='';
  confirmPass = '';
  return("E: Password does not match");
 }
 newPass=newPass.toString().trim();
 if(newPass.length < 6) return("E: Password is not long enough (Minimu
m length: 6 characters)");

 db = that.ODBCCon;
 var sqlQuery = "update [dbo].["+objUserTable.name+"] SET ["+objUserTa
ble.cols.password+"] = '"+newPass+"', ["+objUserTable.cols.firstlogin+
"] = '', ["+objUserTable.cols.modifiedOn+"]=CONVERT (datetime, GETDATE
()),["+objUserTable.cols.modifiedBy+"]='"+USERNAME+"' WHERE "+objUserT
able.cols.user+" = '" + zuser+"'"
 println(sqlQuery);
 try{
  t = db.exec(sqlQuery);
 }
 catch(err){
  message(err.description);
```

```
 }
 println("------> t: ",t);
 //set("V[z*]","");
 set("V[newPass]","");
 set('V[confirmPass]','');
 changeLayout("MODIFY_USER_DETAIL");
 return("S: Password changed successfully");
}
```

Unique solution ID: #1102
Author: Punil Shah
Last update: 2019-09-25 12:12