Database Guide

Document Information

Software Version:	V4.0.3.5
Creation Date:	27 August, 2020
Last Edit Date:	29 April, 2022
Version:	V1.4

Table of Contents

1. Scope

This document details different ways to create connections between ADISRA SmartView and the desired relational database(s).

2. Summary

There are different ways to create a connection between ADISRA SmartView and the databases, so it is important to establish why each connection will be needed and how it is going to be used.

Examples:

- If the user just wants to store tag values in the database, the user will only need to create a global database connection in the top ribbon and use that connection in the Tag History document.
- If the user wants to store the alarm history, it will be a similar solution. The user will need to create a global database connection and use that connection in the Alarm History document.
- If the user needs to execute queries such as “create table”, “select”, “update” or "delete”, the user may use the global database connection and the system function library, or the user can write the entire connection using their own script. In this second example, the user will not need the global database connection.

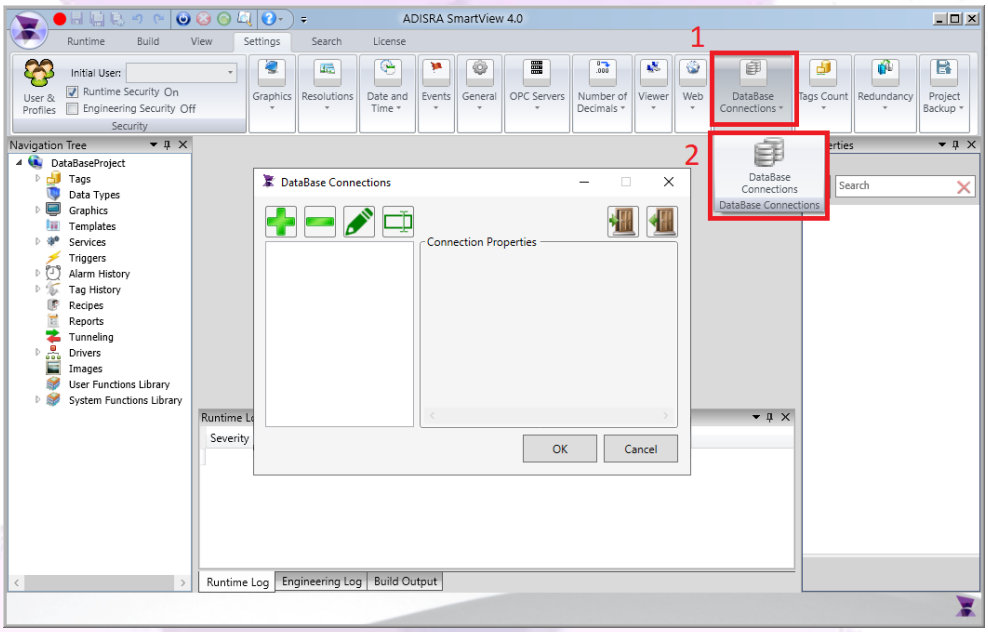
Please review the following sections and feel free to use one of the solutions in an application.

3. Configure Global Database Connection

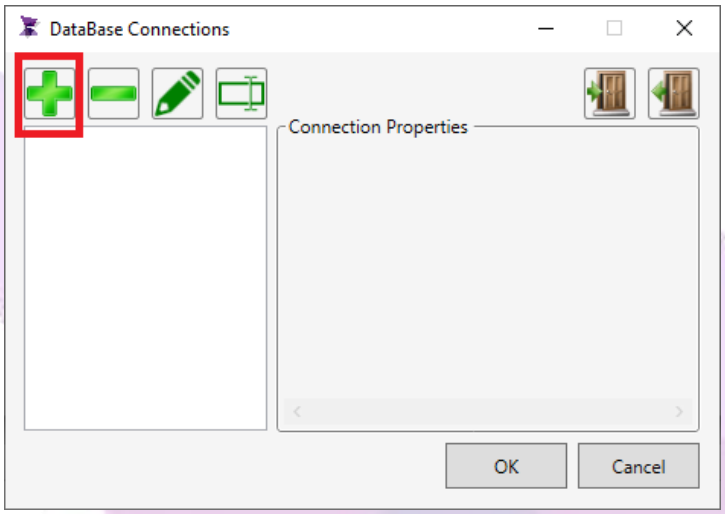
3.1. Create a New Connection

The Global Database Connection is part of the project settings. It allows the user to create a database connection through a connection wizard and use that connection in their application. To configure a new global database connection, please follow the steps below:

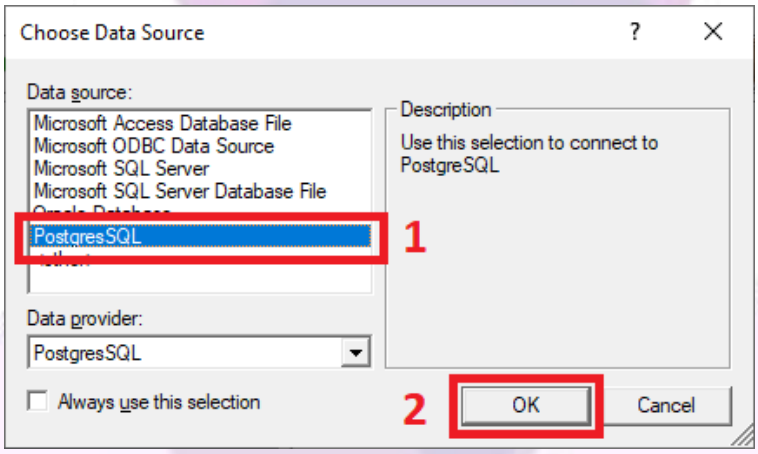
- In the ADISRA SmartView ribbon, go to "DataBase Connections" and click the "DataBase Connections" button, the "DataBase Connections" window will open.



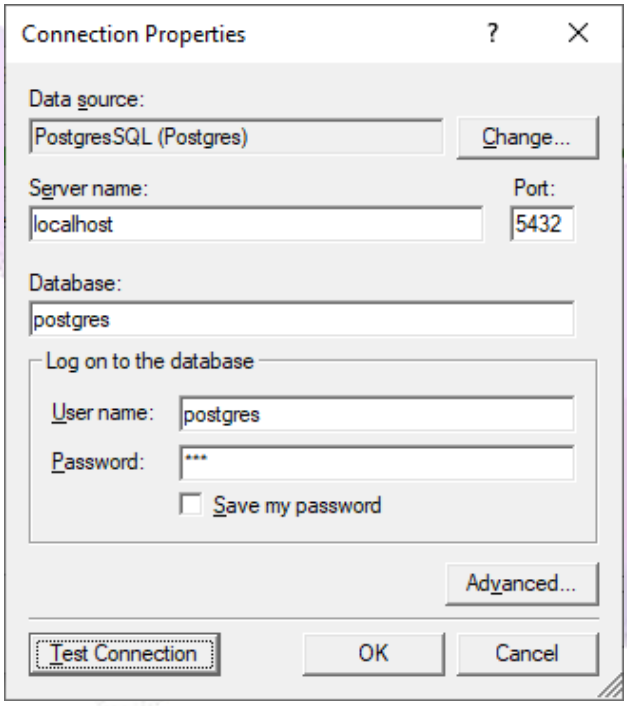
- To add a new database connection, click the "+" button shown in the red box below:



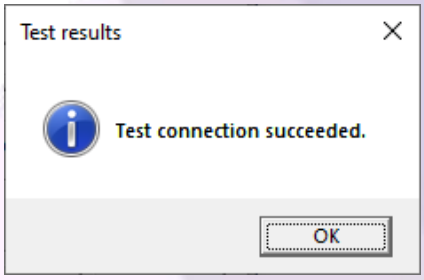
- In this example, we will configure a PostgreSQL connection, select PostgreSQL then click OK:



- The "Connection Properties" window will appear, the user should type in the database information that the user wants to connect. The image below is just an example:

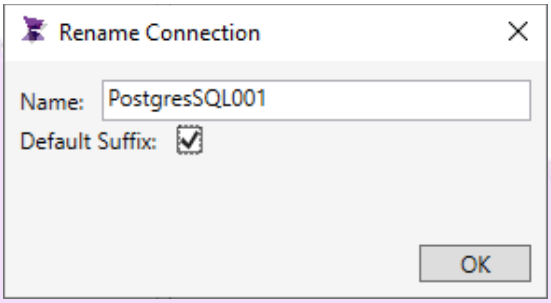


- Click the "Test Connection" button, if all the information is correct, it will display the following dialog:

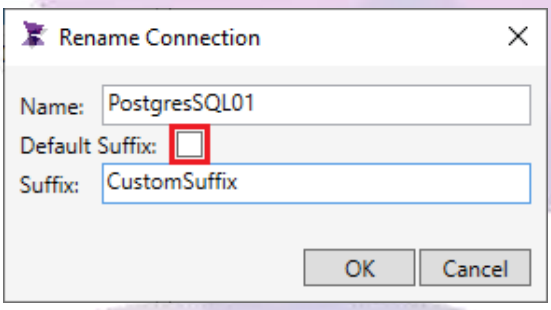


- If the test connection succeeded, please click OK.
- If the test is not successful, then an error message will be displayed with the error description. The error message may state a driver installation is missing or connection information does not match.
- The Advanced button displays custom configurations for the selected database driver. The database being used will determine if the additional information is required.

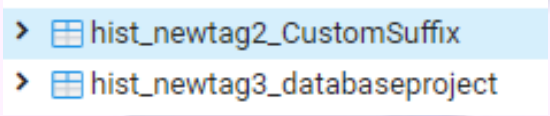
- Click OK on the "Connection Properties" window, then name the connection and select if the tables will or will not be created with the default suffix. The default suffix is the name of the project:



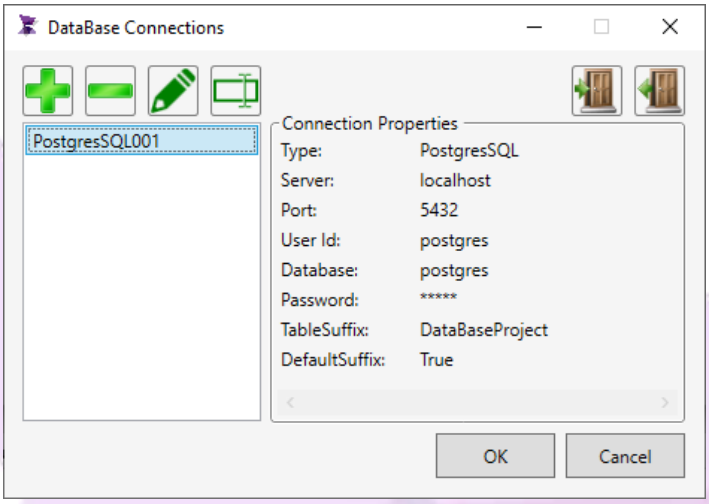
- If the user wants to change the suffix, uncheck the Default Suffix check box and enter the new suffix:



- See the example of two tables created, the first one with a custom suffix, and the second one with the default suffix which is the name of the project. In this example the project name is "databaseproject":



- The user should see the connection created:



- Different Databases will have different options in the "Connection Properties" window. The user should configure it according to the Database configuration. Below are the different options for each Data source:

3.1.1. Microsoft Access**

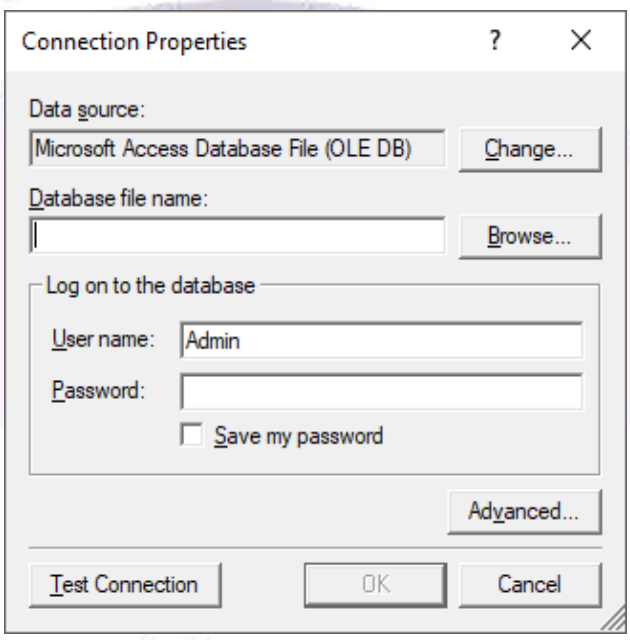


Figure 1 - Microsoft Access Database

3.1.2. Microsoft ODBC

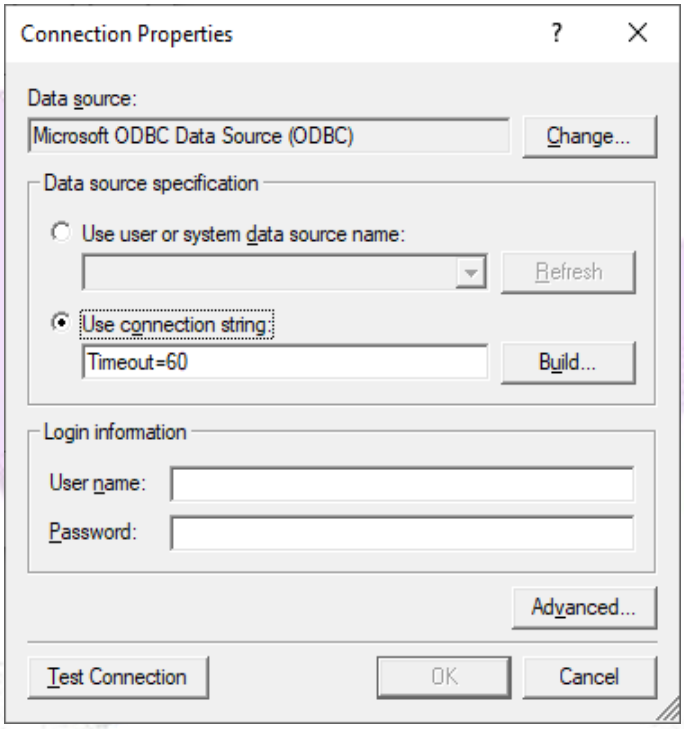


Figure 2 - ODBC Data Source

NOTE: The ODBC database will not be show correctly if the compatibility below is not followed:

- Using Windows 64 Bits
    - ADISRA 64 Bits and ODBC 64 Bits – Works normally.
    - ADISRA 32 Bits and ODBC 32 Bits – Works normally.
- Using Windows 32 Bits
    - ADISRA 32 Bits and ODBC 32 Bits – Works normally.

3.1.3. Microsoft SQL Server

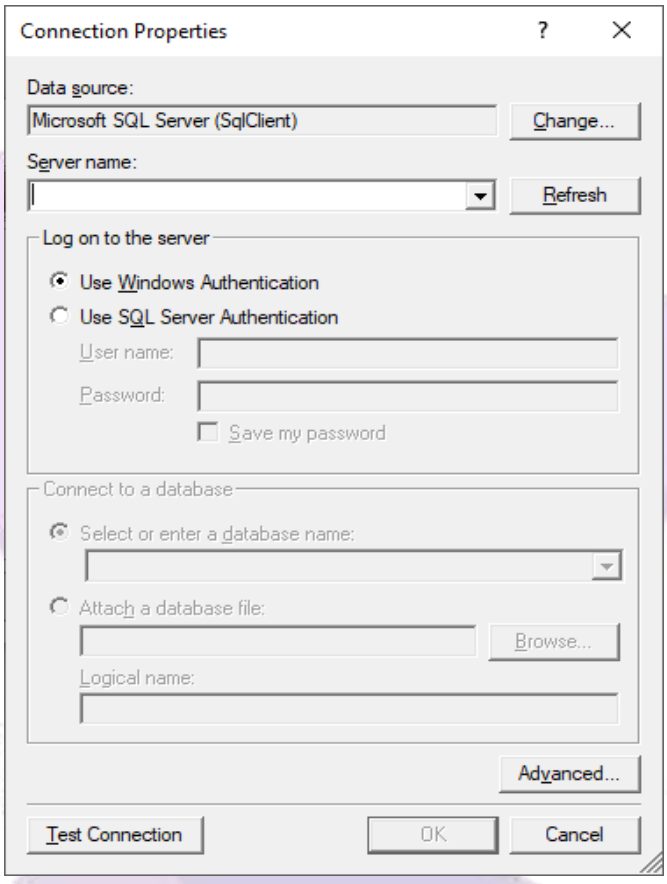


Figure 3 - Microsoft SQL Server Database

3.1.4. Microsoft SQL Server

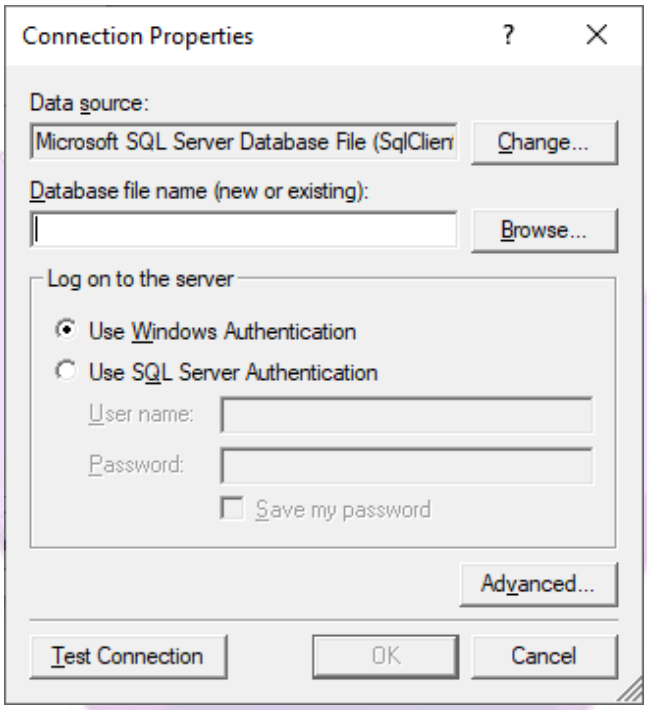


Figure 4 - Microsoft SQL Server Database File

3.1.5. Oracle Database

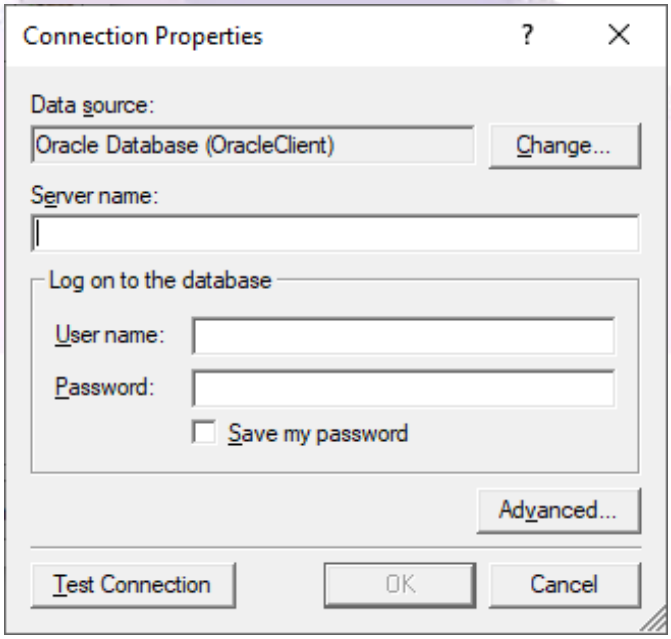


Figure 5 - Oracle Database

3.1.6. PostgreSQL

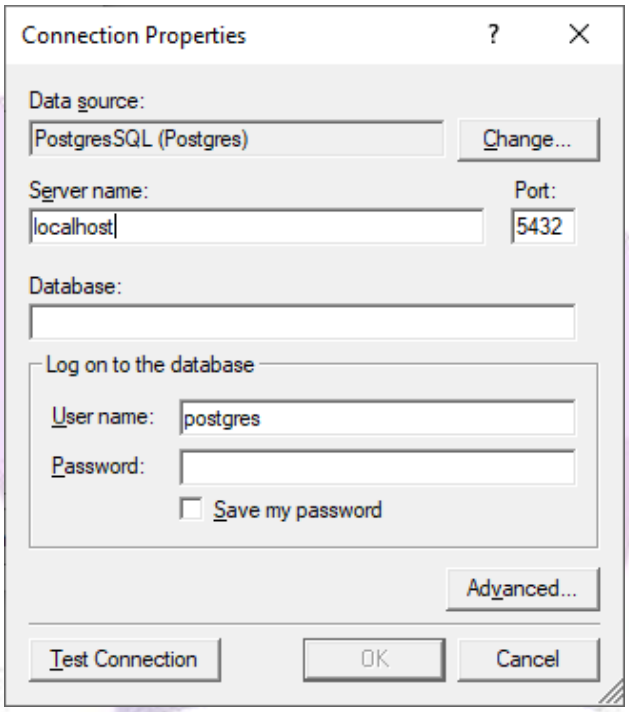
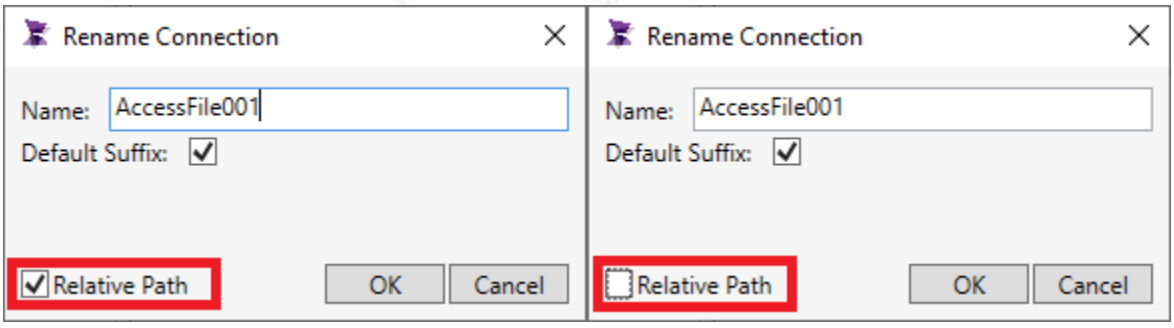


Figure 6 - PostgreSQL Database

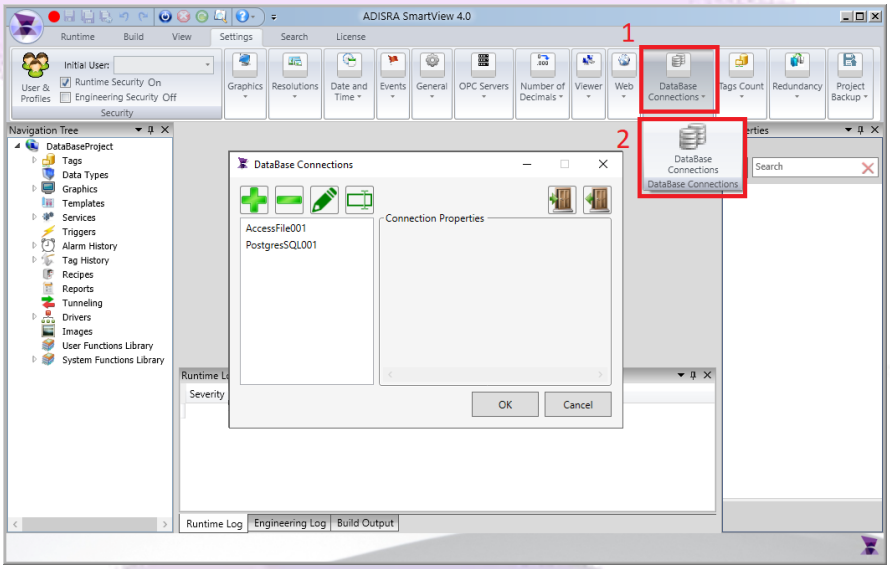
** Microsoft Access - When configuring or renaming connections to an Access database file or a Microsoft SQL Server database file there will be an extra option called “Relative Path”, if unchecked ADISRA SmartView will look for the file in the absolute path of the hard drive. For example, “C:\Users\Documents\AccesDB.accdb”. It is important to understand if the user executes the project in another machine, it will not be able to find the file. If the option “Relative Path” is checked, ADISRA SmartView will look for the file in the relative path from the project path. Selecting the “Relative Path” option will make it easier to export the project to another machine.



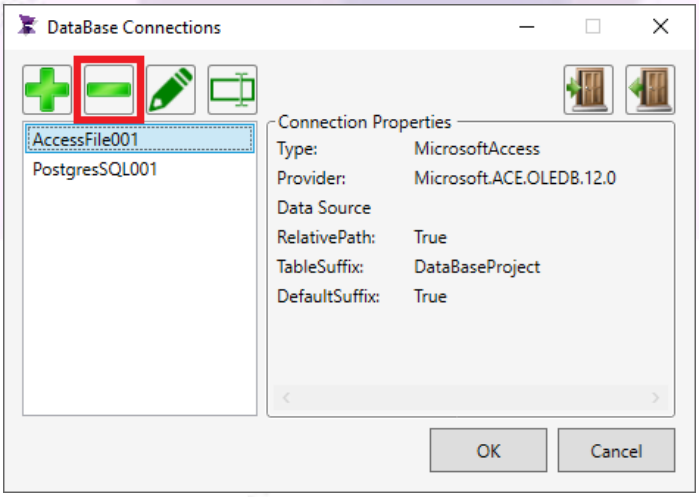
3.2. Remove a Connection

To remove a database connection, follow the steps below:

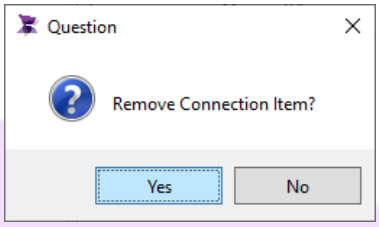
- In the ADISRA SmartView ribbon, go to “DataBase Connections” and click the “DataBase Connections” button, the window “DataBase Connections” will open.



- Select the connection the user wants to remove and click the “-” button shown in the red box below.



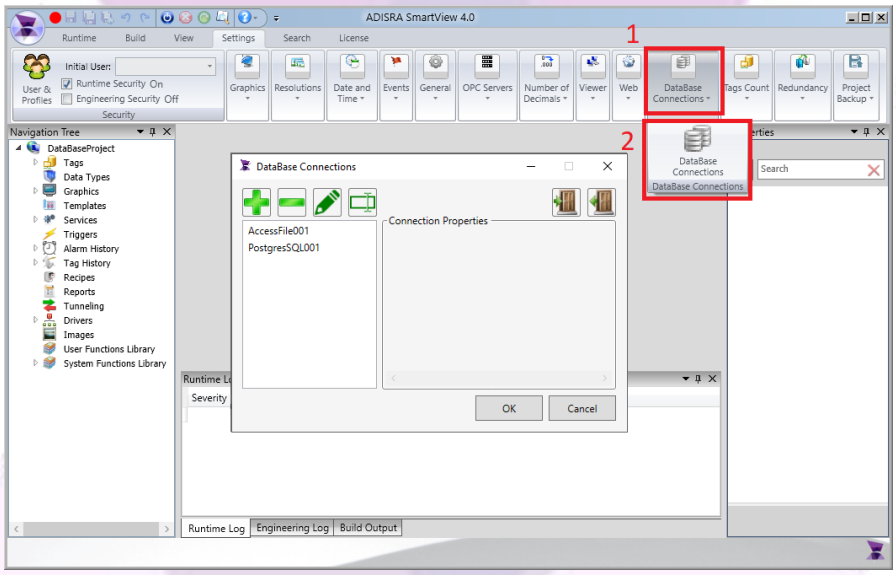
- Confirm the user wants to remove the connection item.



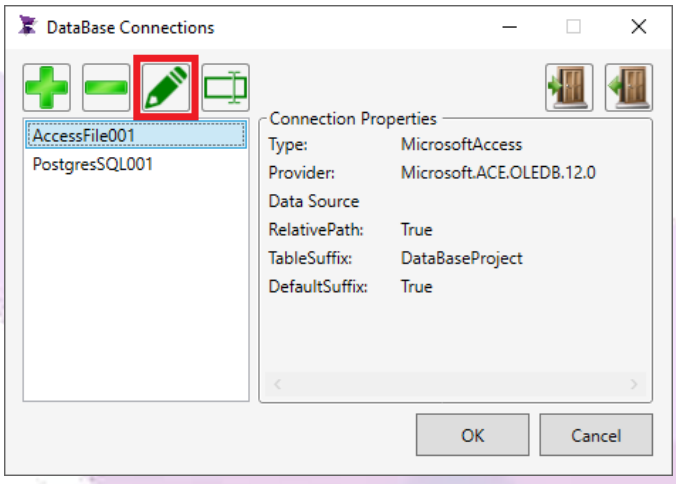
3.3. Edit a Connection

To edit a database connection, follow the steps below:

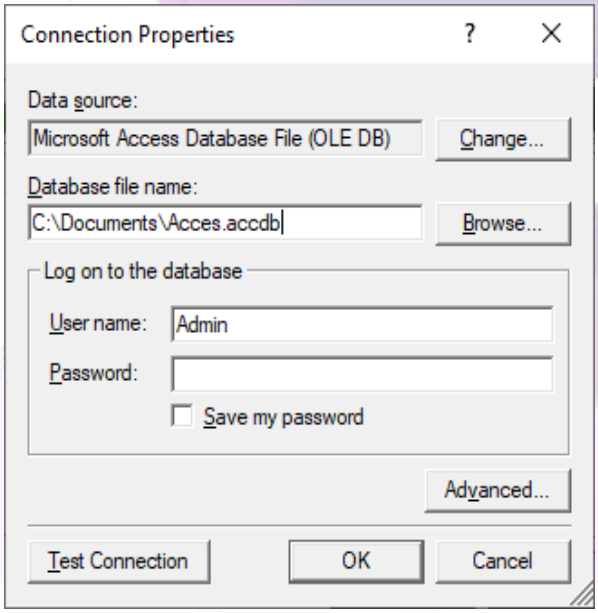
- In the ADISRA SmartView ribbon, go to “DataBase Connections” and click the “DataBase Connections” button. The window “DataBase Connections” will open.



- Select the connection the user wants to edit and click the “pencil symbol” button shown in the red box below.



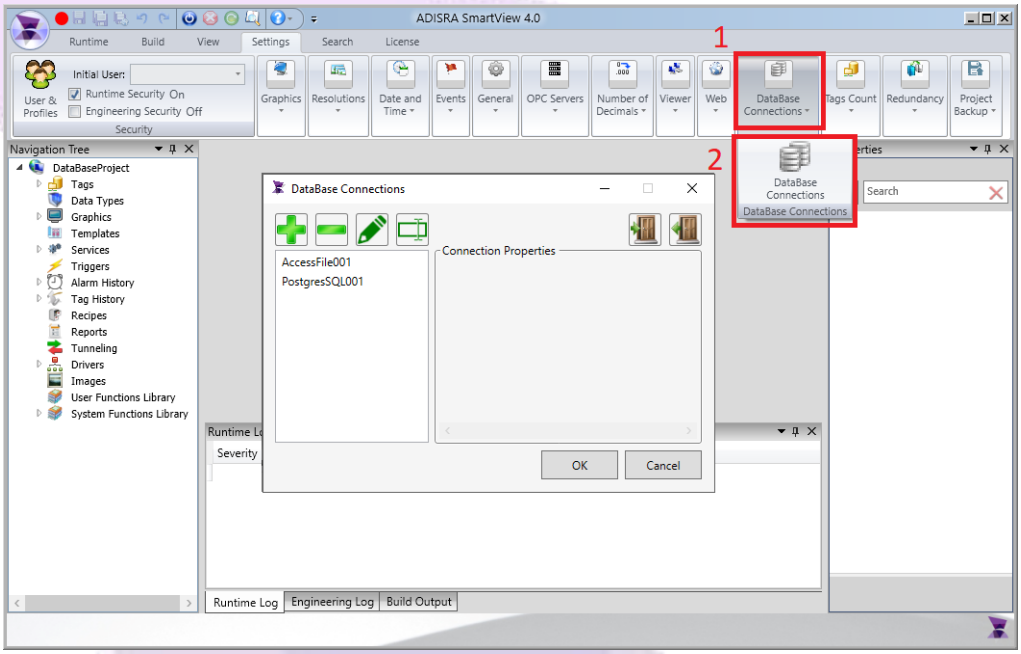
- It will open the “Connection Properties” window so the user can edit the connection.



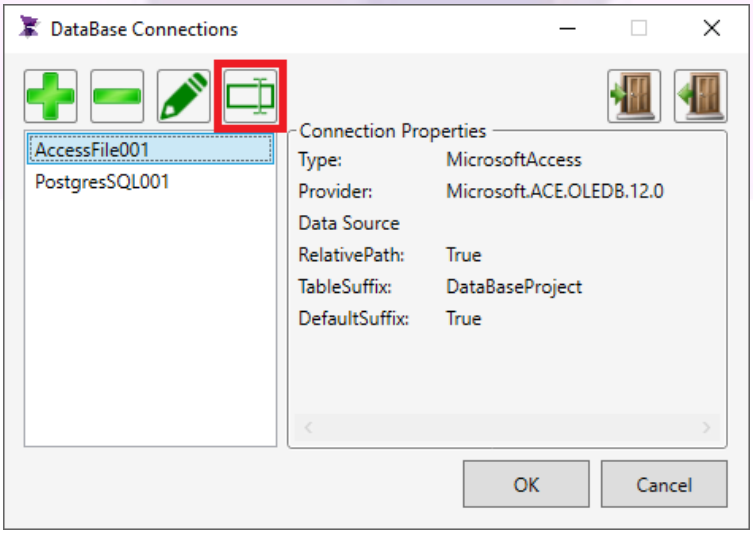
3.4. Rename a Connection

To rename a database connection, follow the steps below:

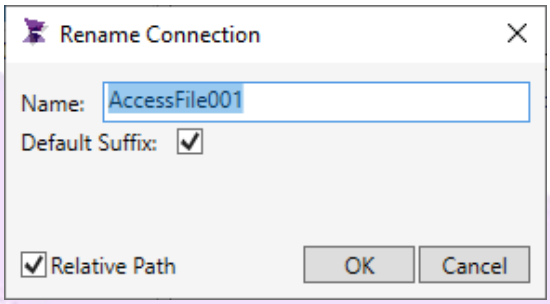
- In the ADISRA SmartView ribbon, go to “DataBase Connections” and click the “DataBase Connections” button. The window “DataBase Connections” will open.



- Select the connection the user wants to rename and click the “Rename Connection” button shown in the red box below.



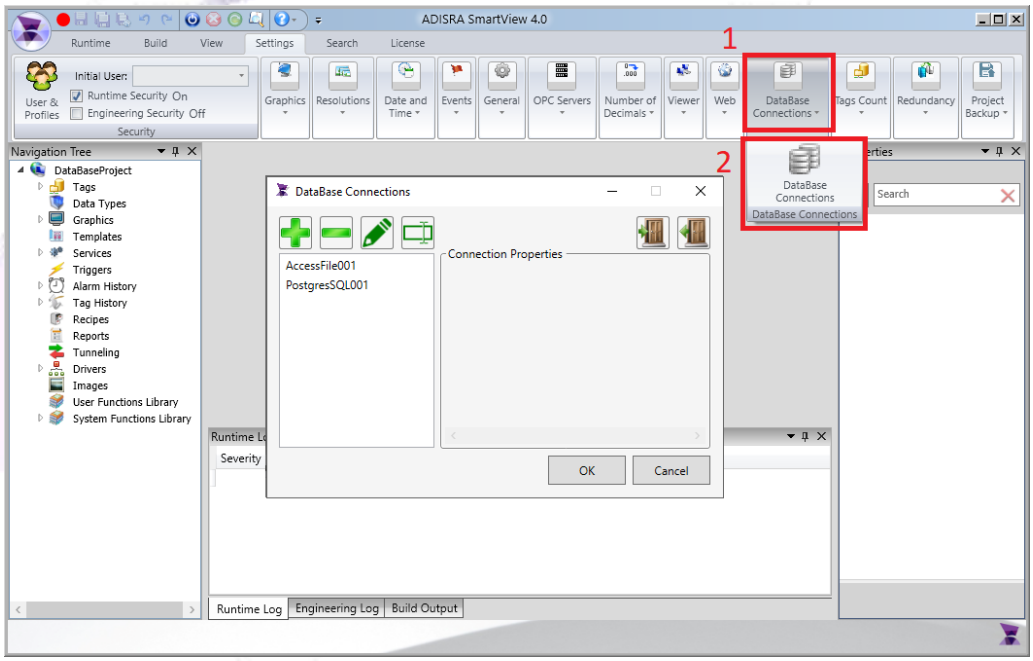
- The "Rename Connection" window will open and the user can rename the connection. Select "OK" to complete the renaming.



3.5. Import and Export a Database Connections

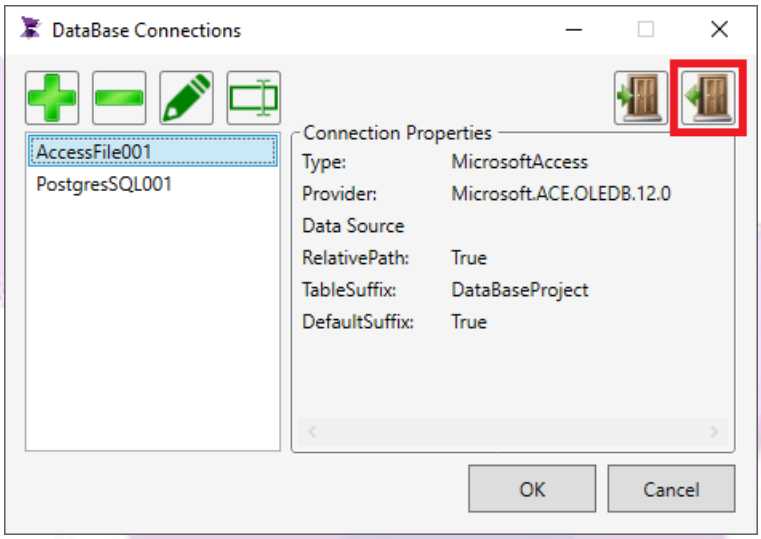
To import and export configured database connections, follow the steps below:

- In the ADISRA SmartView ribbon, go to "DataBase Connections" and click the "DataBase Connections" button. The window "DataBase Connections" will open.

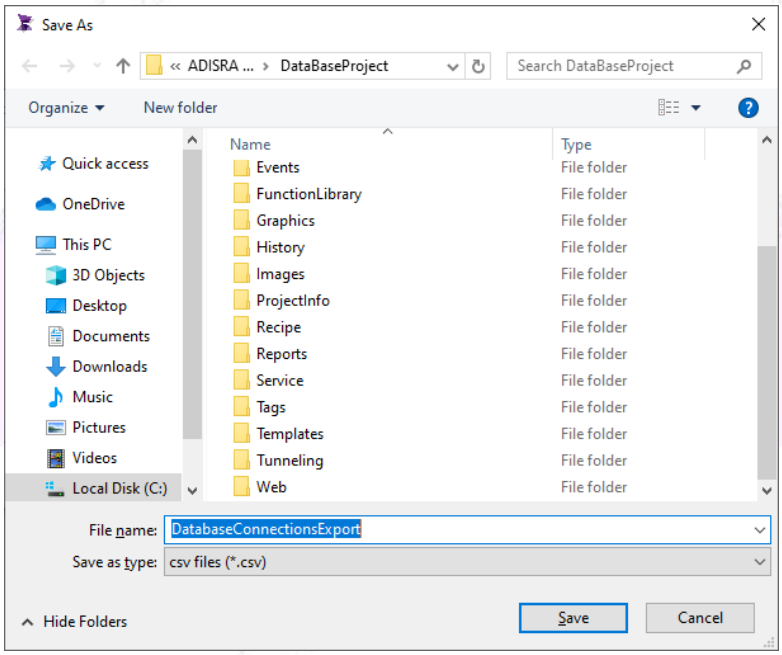


3.5.1. Export Database Connections

- Click the "Export Connections" button shown in the red box below.

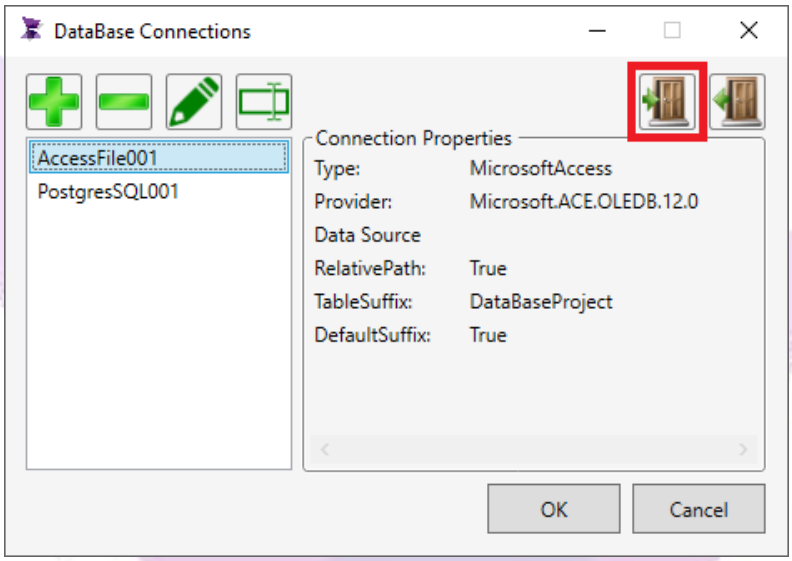


- When exporting the connections, a new dialog box will display. The user has the option to select the folder and insert a name for the exported file. Click the "Save" button and the file will be generated.

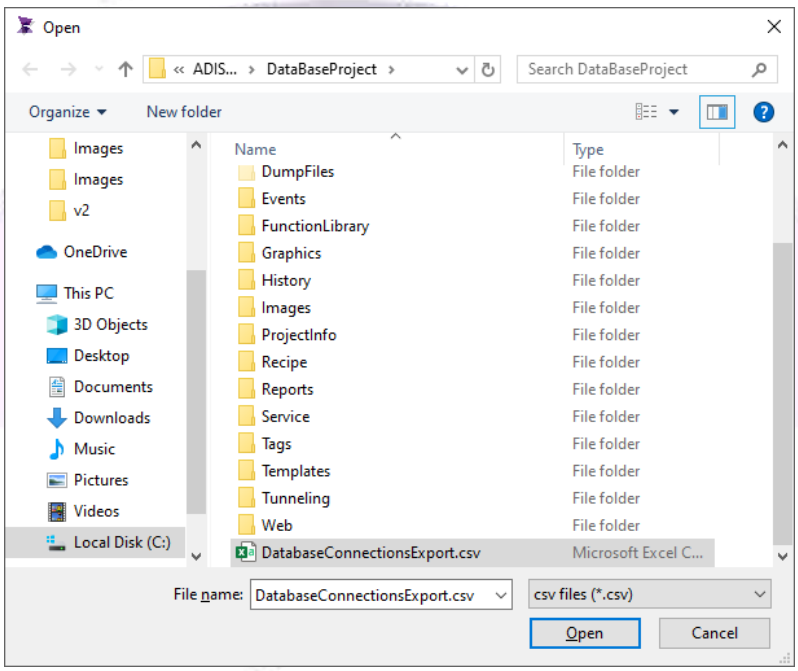


3.5.2. Import Database Connections

- Click the "Import Connections" button shown in the red box below.

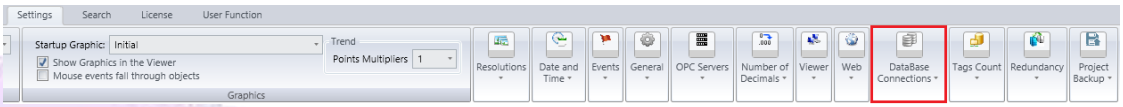


- After clicking the "Import Connections" button, a dialog box will display allowing the user to select a csv. file previously generated by ADISRA SmartView.



- Select the exported file, name the file and click the "Open" button to import the file into the current project. The "Import Connection" functionality allows the user to import a previously exported file.

4. Global Database Connections



The "Global Database Connection" option, highlighted in the red box above, can be used to store Tag Values and Alarms in the database. The option can be used to run queries using the SVDatabase System function Library.

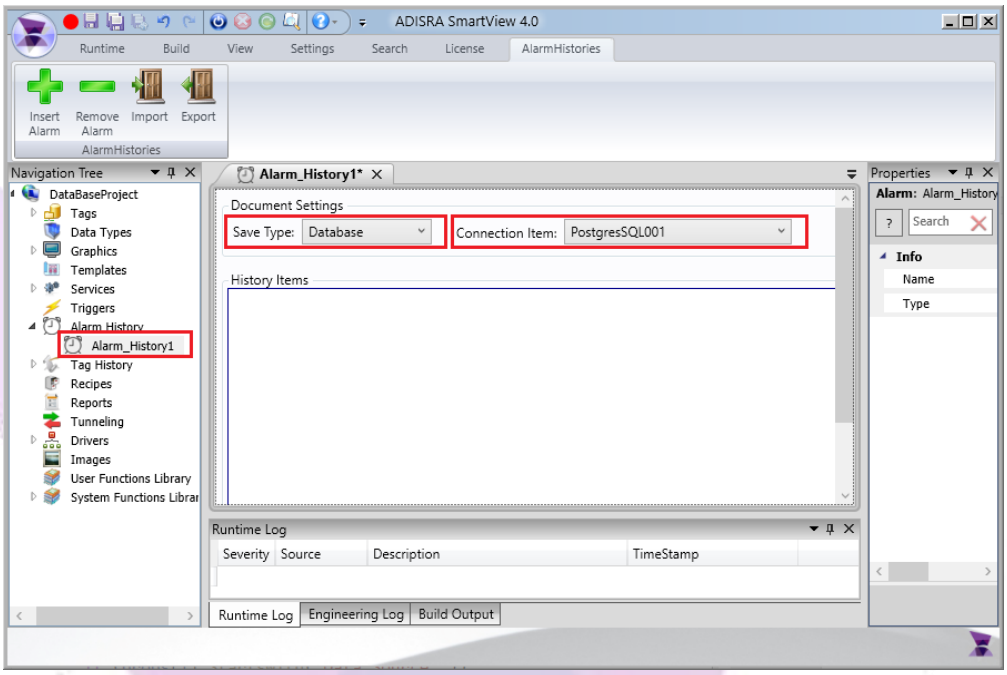
To use the configured global database connections, follow the steps below.

4.1. Using Database Connections with Alarm History

As soon as the runtime is started, the alarm history table will be created by ADISRA SmartView with all the needed columns. Each alarm history document will generate one table in the database. The database history table created will be one table per document with all tags configured in it.

By contrast, in Tag's History option, each the tags inside the document will generate a new table.

- The alarm history can be saved into a configured database. Open the alarm history document the user wants to configure on the "Save Type" combo box, select "Database" and on the "Connection Item" combo box, select a configured "Database Connection".



- As soon as the runtime is executed, the alarms configured in the Alarm History document will be saved to the database. The values saved to the database can be loaded by the Alarm object as shown in the example below.

Tag Name	Tag Description	Group	Priority	Start Time	Return Time	Type	Message
TagInt		Alarm_History1	0	08/28/2020 06:24:43 PM	08/28/2020 06:24:53 PM	Hi	
TagInt		Alarm_History1	0	08/28/2020 06:24:43 PM		Hi	
TagInt		Alarm_History1	0	08/28/2020 06:24:13 PM	08/28/2020 06:24:23 PM	Lo	
TagInt		Alarm_History1	0	08/28/2020 06:24:13 PM		Lo	
TagInt		Alarm_History1	0	08/28/2020 06:23:43 PM	08/28/2020 06:23:53 PM	Hi	
TagInt		Alarm_History1	0	08/28/2020 06:23:43 PM		Hi	

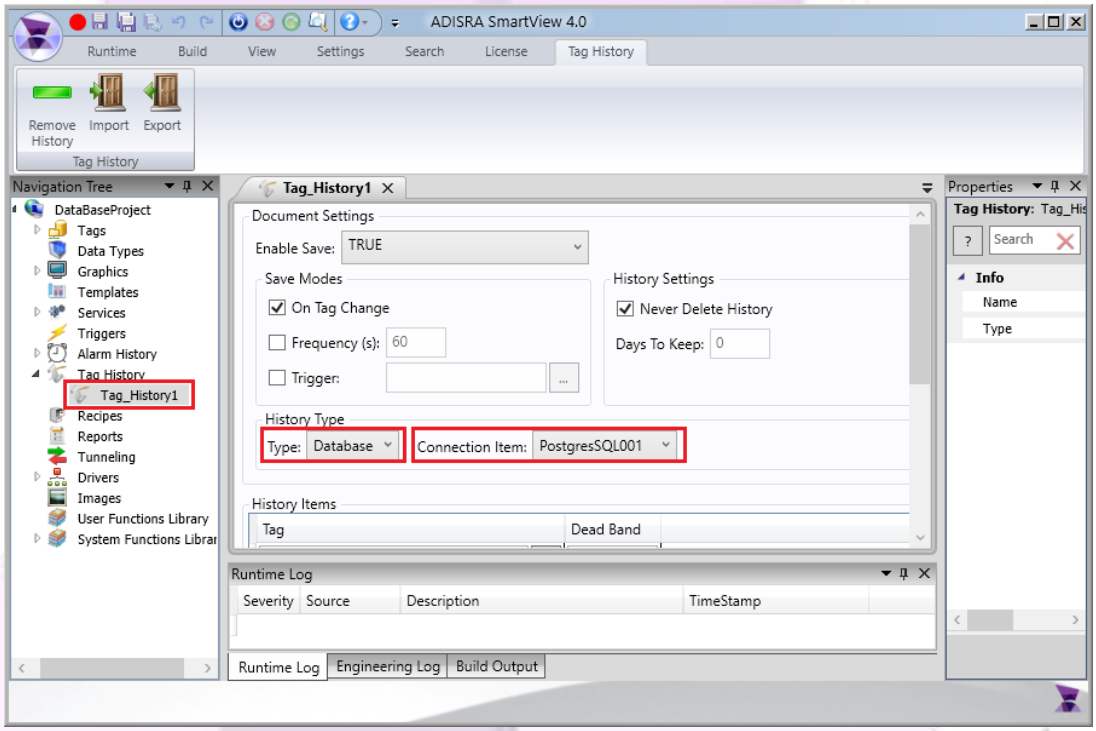
- The following image shows the database table created. The table is created per document configured. So it will contain each tag configured in the document and its relevant alarm information.

	id [PK] integer	tagname character (255)	alarmstate integer	lastalarmstate integer	type character (255)	priorityint integer	messagestring character (255)	groupstring character (255)	ts numeric	acktimedate numeric	returntimedate numeric	tagdescription character (255)
1	0	TagInt	1	0	Hi	0		Alarm_History1	358235264867	0	0	
2	1	TagInt	1	0	Lo	0		Alarm_History1	358535568182	0	0	
3	2	TagInt	1	2	Hi	0		Alarm_History1	358835733067	0	0	

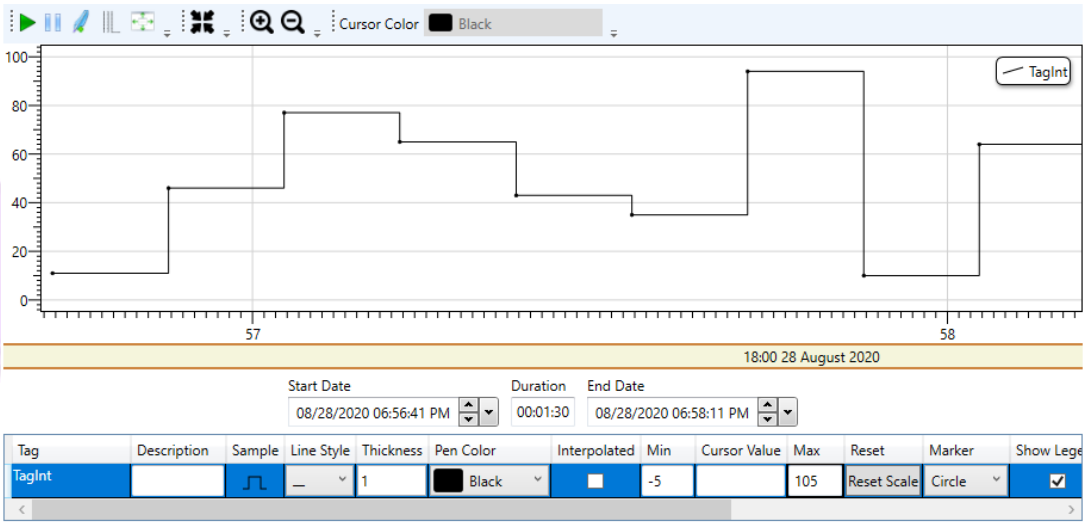
4.2. Using database Connections with Tag History

As soon as the runtime is started, the tag history table will be created by ADISRA SmartView with all the needed columns. It will be one table per tag.

- The Tag History can be saved into a configured database. Open the Tag History document the user wants to configure. Inside the "History Type" area in the "Type" combo box, select "Database"; and in the "Connection Item" combo box, that will appear after selecting the "Type" as "Database", select a configured "Database Connection".



- As soon as the runtime is executed, the tags configured in the Tag History document will be saved to the database. The values saved to the database can be loaded by the Trend (History) object as shown in the example below.



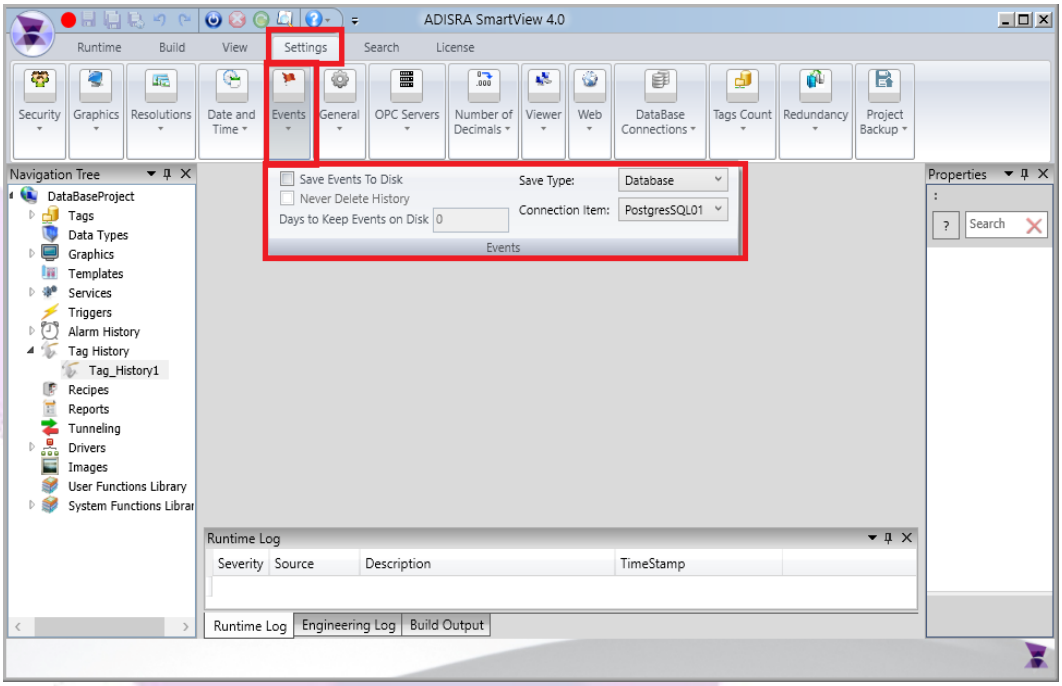
- The following image shows the database table created. It contains the tag value, quality, and timestamp.

	id [PK] integer	tagvalue character (255)	quality integer	ts numeric
1	1	11	192	486026581082
2	2	46	192	486126644657
3	3	77	192	486226655044
4	4	65	192	486326817890
5	5	43	192	486426967352
6	6	32	192	487127281264
7	7	98	192	487227293825
8	8	68	192	487327298629
9	9	65	192	487427452049

4.3. Using Database Connections with Events

It is possible to save the Events into the database. Please follow the steps below to configure it.

- The Events can be saved into a configured database. Select Events in the Settings ribbon; in the “Save Type” combo box, select “Database” and in the “Connection Item” combo box, select a configured “Database Connection”.



- Below is an example of the table created in the Database:

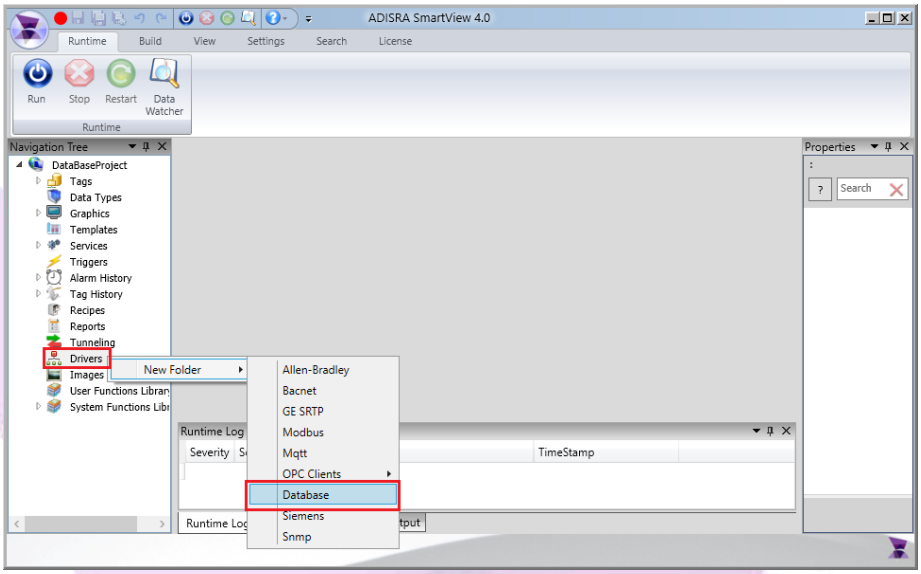
	id [PK] integer	message character (255)	priority integer	gp character (255)	ts numeric
1	0	Message test	0	Group1	776798835303

4.4. Using Database Connections with a Database Driver

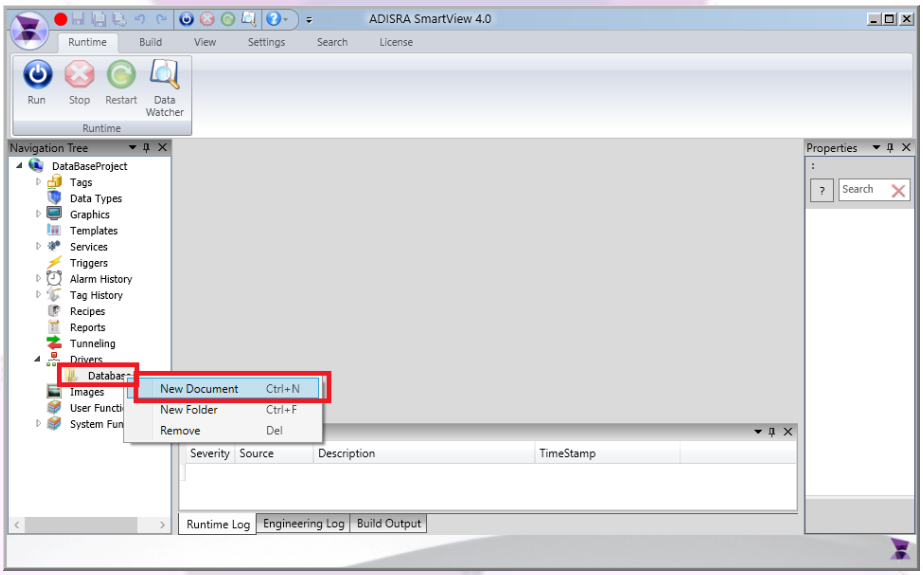
The user can also configure a Database Connection similar to a driver communication. In this section, the user will learn to link a tag with a database record.

- With the Database document in ADISRA SmartView, the user can configure a communication between one or more tags to a selected Database.

- Create a Database document by left clicking the “Divers” node and right clicking the “Database” option inside the “New Folder” as shown in the red box below.



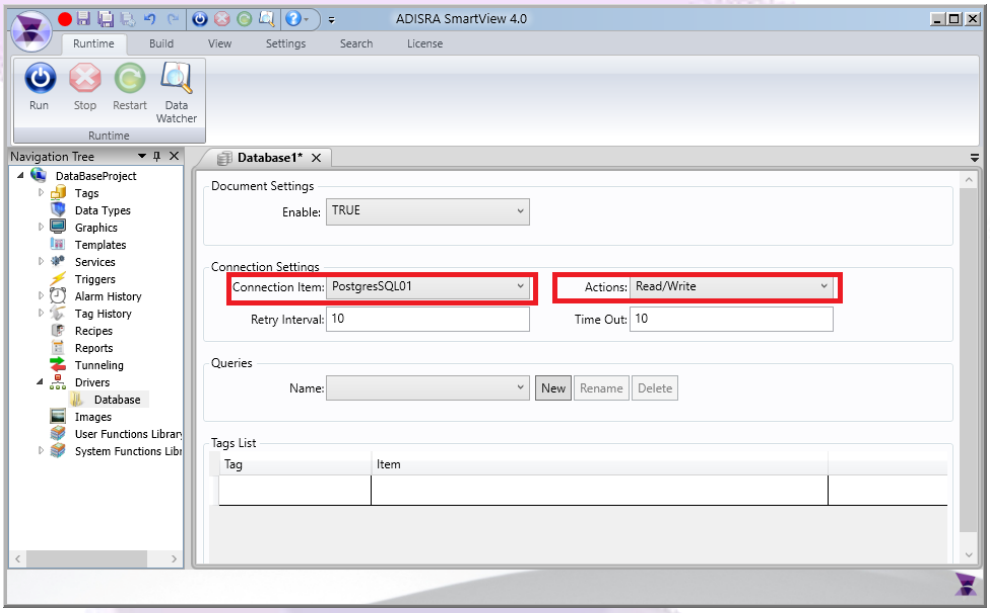
- Left click the newly created "Database" folder and then right click the "New Document" option. A database window opens.



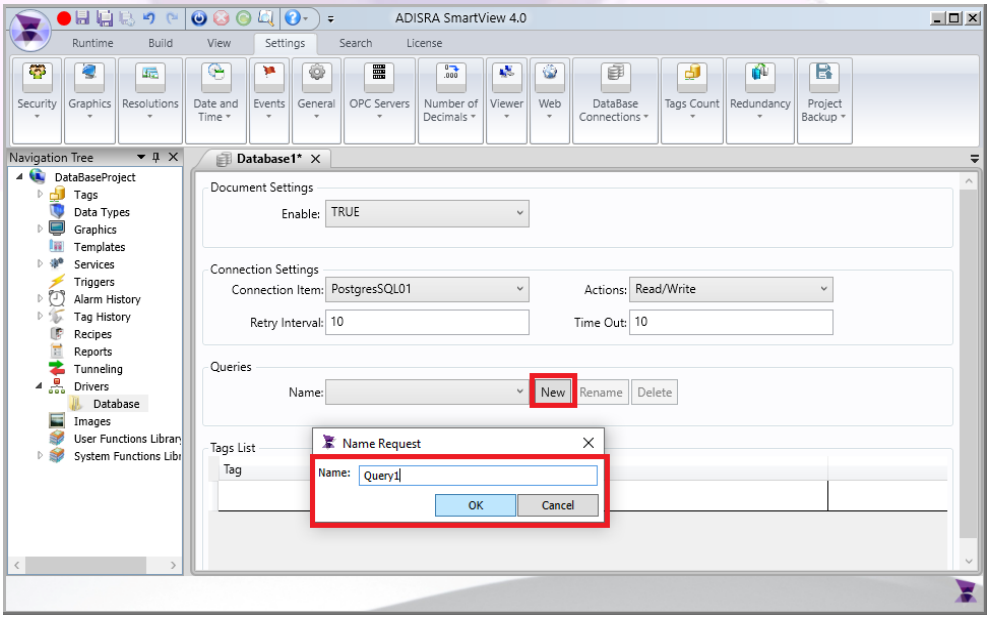
- In the "Connection Item" box, select a configured connection and in the "Actions" box, select the directional flow of the data.

Read	The data only flows from Database Server to Database Client. The Database Client can only read the data from the Database Server
Read/Write	The data flows in both directions, from the Server to the Client and from the Client to the Server. The

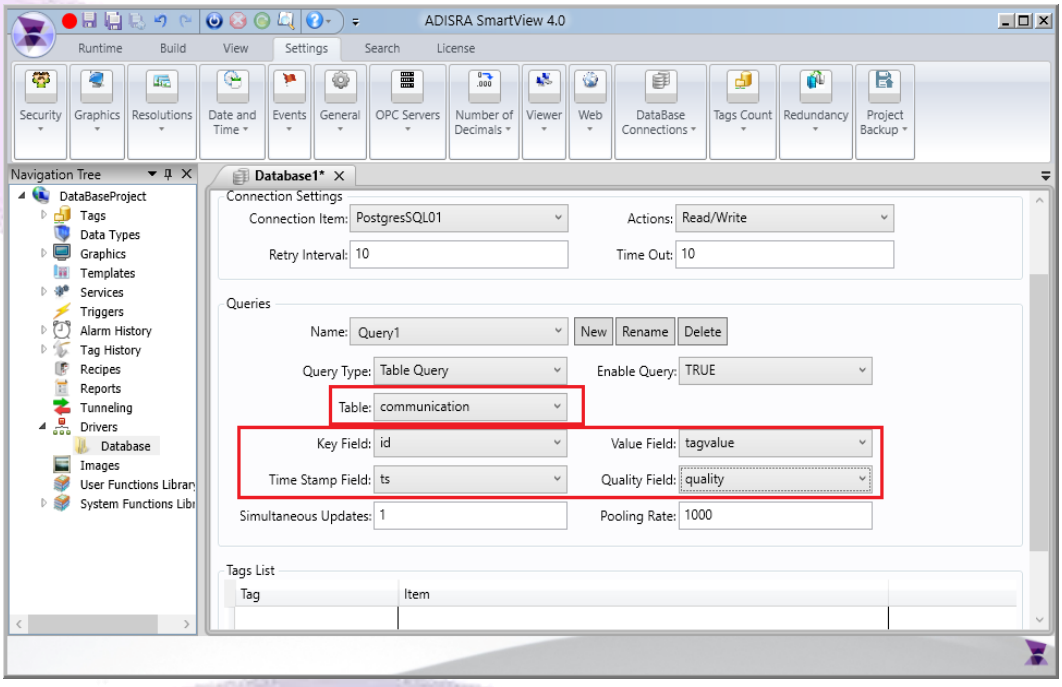
	Database Client can read and write data from the Database Server
Write	The data flows only from Database Client to Database Server. The Database Client can only write data to the Database Server



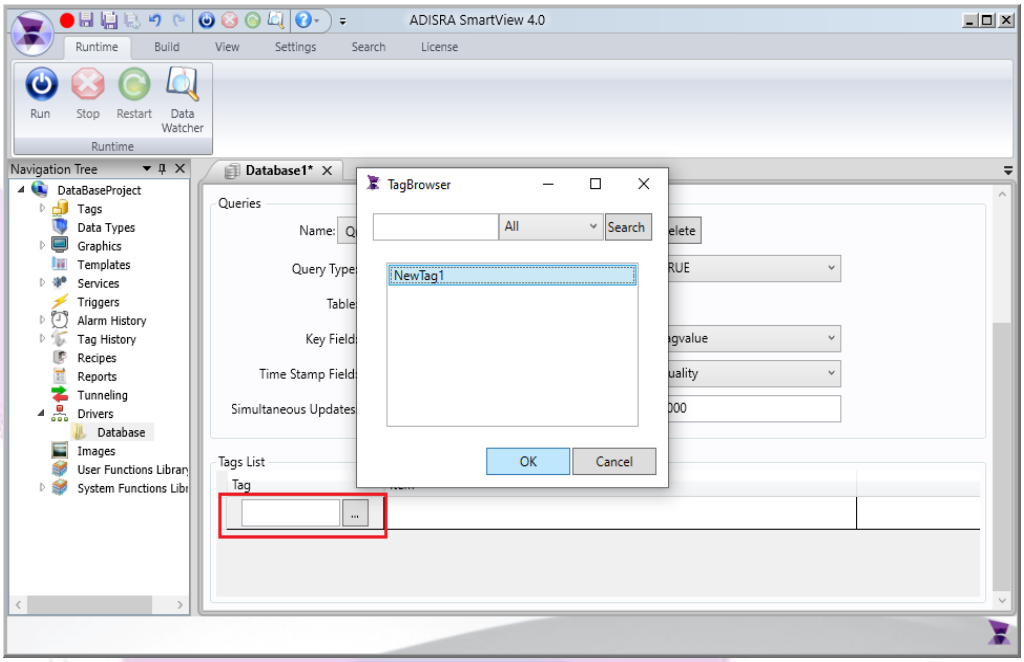
- In the "Queries" area click the "New" button, name the query, and click "OK".



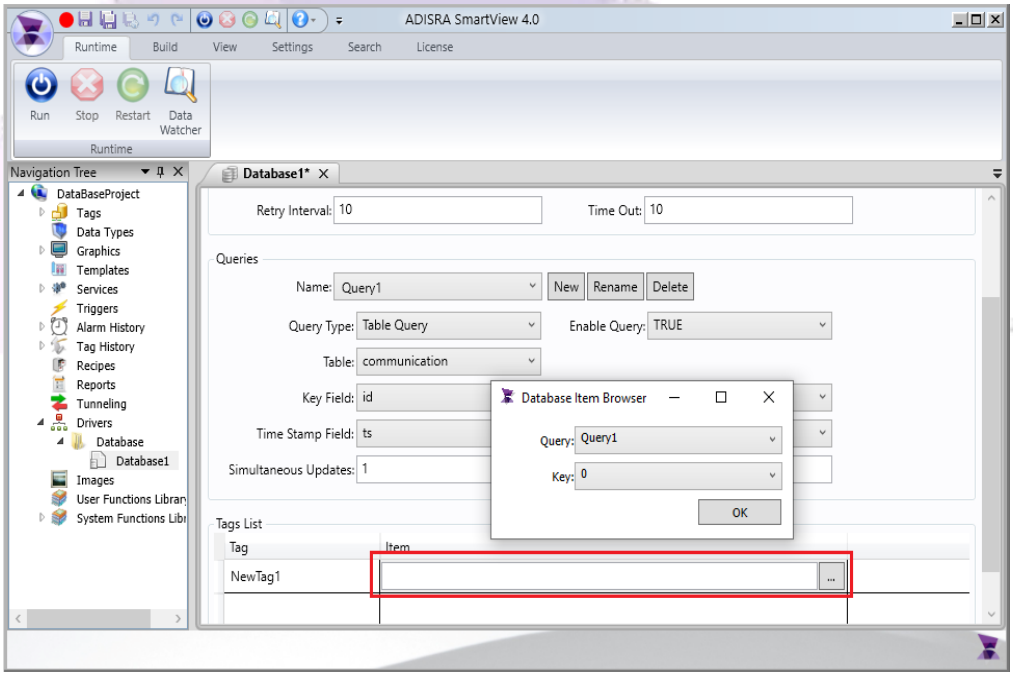
- In the “Table” combo box, select the configured table from the database. In the “Key Field”, “Value Field”, “Time Stamp Field” and “Quality Field” boxes, select the columns configured from the table previously selected.



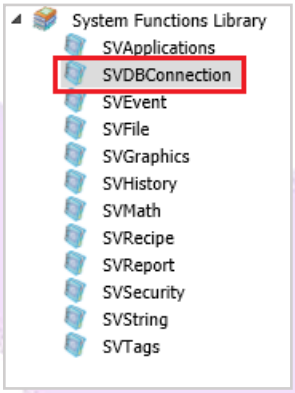
- Inside the “Tags List” area, double click on the cell in the Tag column (as shown by the red box in the image below), then click the “…” button; this action will open the “TagBrowser” window for the user to choose a tag.



- Inside the Tags List area, double click on the cell in the Item column (as shown by the red box in the image below), then click the "…" button; this action will open the "Database Item Browser" window for the user to choose an item from the database to be associated with the selected tag. Select a "Query" created and a "Key" from the database table, click "OK" to save the document.

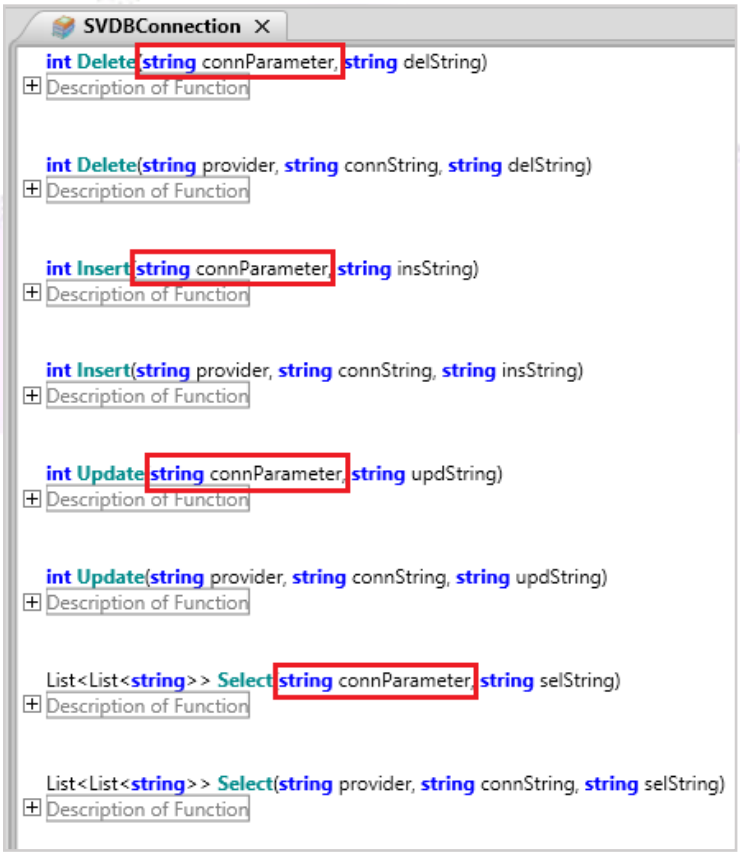


4.5. Using SVDBConnection Functions



The following examples will show how to query a database using the "Global Database Connection" and the "SVDBConnection" that is located within the "Systems Function Library".

A separate chapter will describe different ways to use the "SVDBConnection" to query the database without using the "Global Database Connection". Using the "Database Connection" approach is a simpler more direct approach to database query.

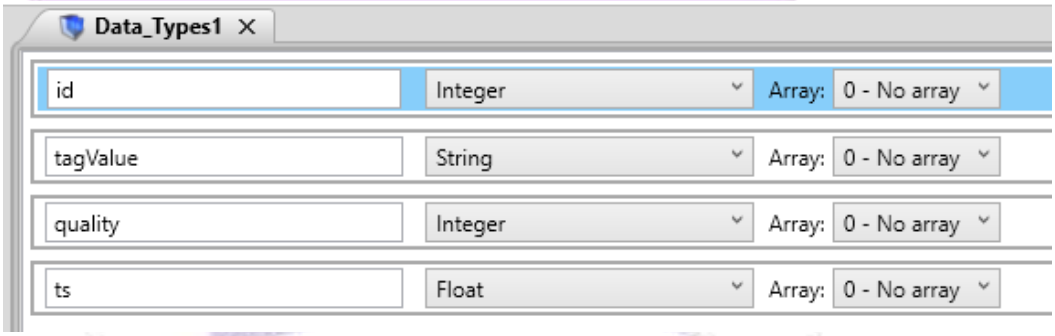


NOTE: The highlighted functions above contain an input parameter called "connParameter" which is the name of the global database connection.

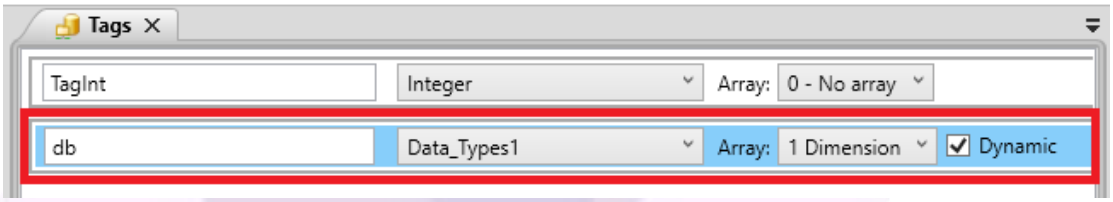
4.5.1. Using SVDBConnection.Select() function

- The example below shows a script that will read values from a table in the database and add those values to existing tags.

  The Data Type configured:



- The Tag configured:



- The table in the database:

	id [PK] integer	tagvalue character (255)	quality integer	ts integer
1	0	400	0	1234654
2	1	Name01	1	25646
3	2	N1	2	56465

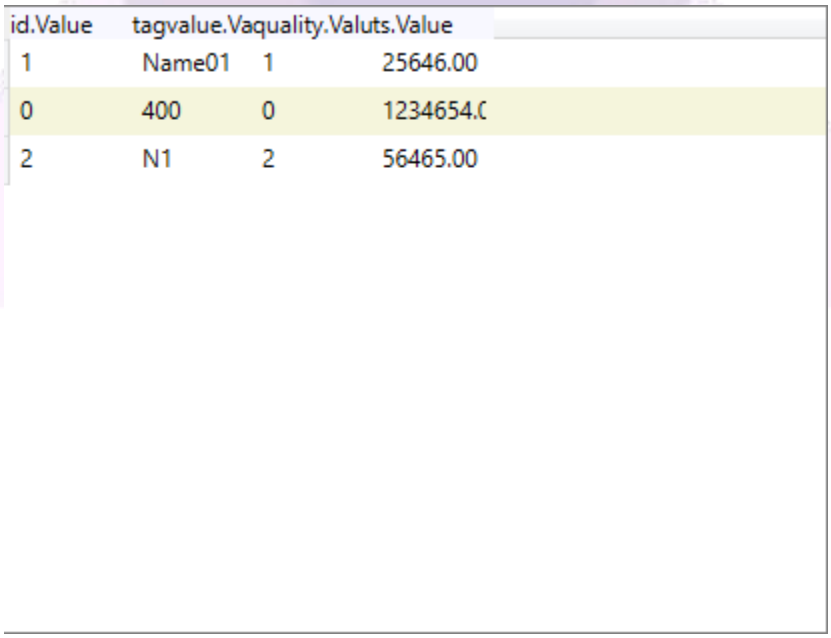
- Now add the script below to a button or when a screen opens to execute it.

```
List<List<string>> selection = new List<List<string>>();
selection = SVDBConnection.Select("PostgresSQL001", "select id, tagvalue, quality, ts FROM public.functions;");

if (selection != null){
    foreach (List<string> row in selection){
        string rowString = "";

        System.Collections.Generic.Dictionary<string, object> dic = new System.Collections.Generic.Dictionary<string, object>();
        dic.Add("id.Value", row[0]);
        dic.Add("tagValue.Value", row[1]);
        dic.Add("quality.Value", row[2]);
        dic.Add("ts.Value", row[3]);

        SVTags.AddDynamicTag("db", dic);
    }
}
else {
    SVApplications.Output("Error occurred during query");
}
```

- The script will create a list to store all the database values and then it will start a loop to add each one of them to the dynamic tag.

NOTE: The first parameter of SVDBConnection.Select is the global database name, in this example, "PostgresSQL001".

- The image below shows the result after the script is executed in a MultiTagViewer object.



4.5.2. Using SVDBConnection.Insert() function

- The example below shows how to create a script that will insert a new table entry in the database.

- The image below shows the table in the database before the script is executed

	id [PK] integer	tagvalue character (255)	quality integer	ts numeric
1	0	3 ...	192	110419356292

- The script configured:

```
SVDBConnection.Insert("PostgresSQL001", "INSERT INTO public.functions(id, tagvalue, quality, ts) VALUES("1,1,192,637304110419356292)");
```

NOTE: The first parameter of SVDBConnection.Insert is the global database name, In this example, "PostgresSQL001".

- The table below shows its values after the script is executed:

	id [PK] integer	tagvalue character (255)	quality integer	ts integer
1	0	3	192	110419356292
2	1	1	192	637304110419356292

4.5.3. Using SVDBConnection.Update() function

- The example shows how to create a script that will alter values in a table in the database.

- The image below shows the table in the database before the script is executed:

	id [PK] integer	tagvalue character (255)	quality integer	ts integer
1	0	3	192	110419356292
2	1	1	192	637304110419356292

- The script configured:

```
SVDBConnection.Update("PostgresSQL001","UPDATE public.functions SET id=1, tagvalue=2 WHERE id=1;");
```

NOTE: The first parameter of SVDBConnection.Update is the global database name, In this example, "PostgresSQL001".

- The table in the database after the script is executed:

	id [PK] integer	tagvalue character (255)		quality integer	ts numeric
1	0	3	...	192	110419356292
2	1	2	...	192	110419356292

4.5.4. Using SVDBConnection.Delete() function

- This example shows how to configure a script that deletes lines from the table in the database.

- The table in the database before the script is executed:

	id [PK] integer	tagvalue character (255)		quality integer	ts numeric
1	0	3	...	192	110419356292
2	1	2	...	192	110419356292

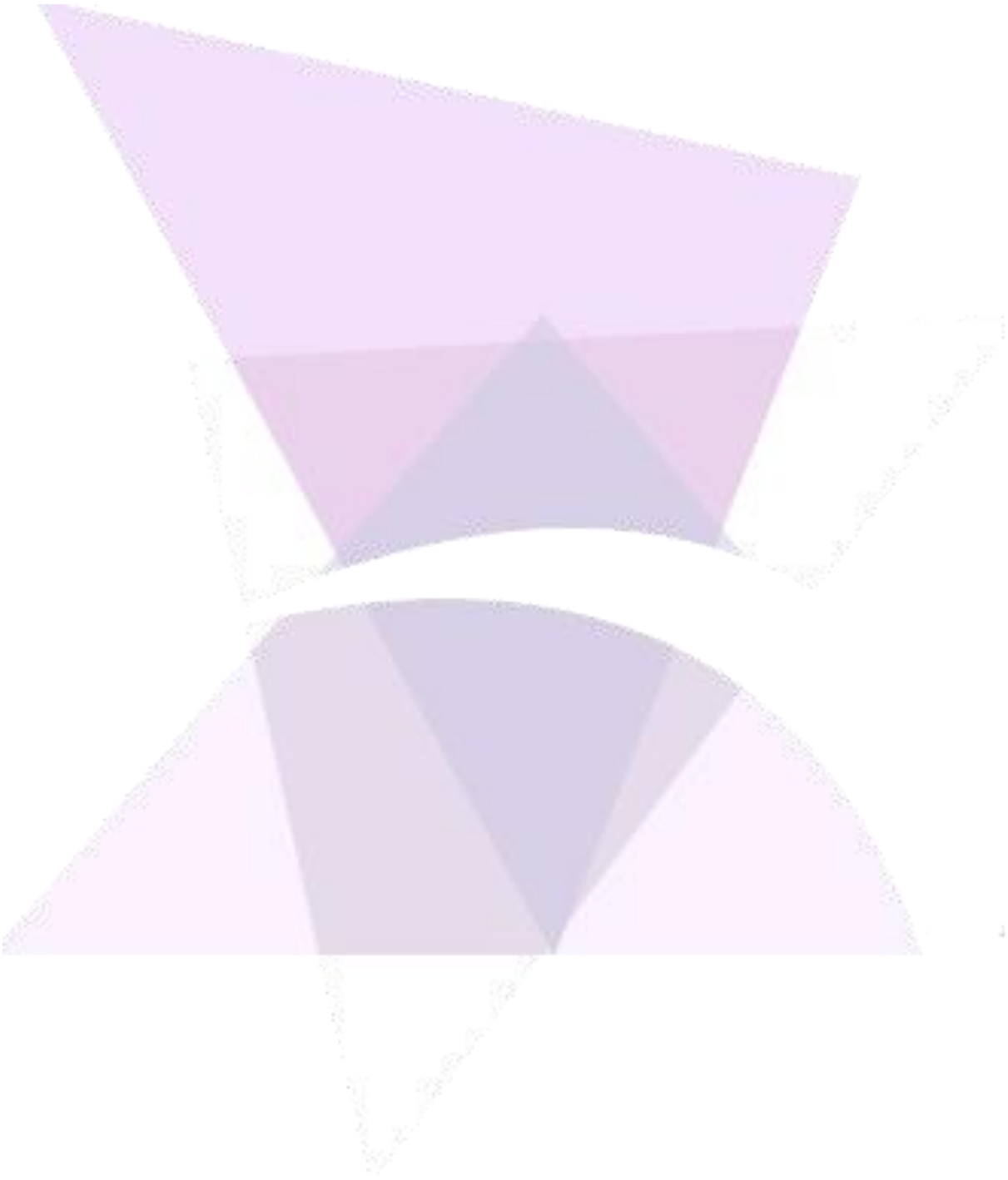
- The script configured:

```
SVDBConnection.Delete("PostgresSQL001","DELETE FROM public.functions WHERE id=1;");
```

NOTE: The first parameter of SVDBConnection.Delete is the global database name, In this example, "PostgresSQL001".

- The table in the database after the script is executed:

	id [PK] integer	tagvalue character (255)	quality integer	ts numeric
1	0	3 ...	192	110419356292



5. Script Database Connection

Scripts can be written inside buttons, a user function, a service, and a trigger. In this chapter the user will be shown different ways to query the database **without** using the "Global Database Connection".

5.1. Using .NET Data Provider

In this example, the script will connect with a PostgreSQL database to add a table into the chosen schema. The example below can only be used to connect to a PostgreSQL database since each database type will have their own parameters to connect.

- The following example creates a table called inside the PostgreSQL database.

```
void CreateEventsTable(string schemaName, bool dropTableIfExists)
{
    string tableName = schemaName + ".events";
    SVApplications.Output("Creating events table:"+ tableName);
    var cs ="Server = localhost; Port=5433; Username = postgres; Password = Postgres1!; Database= adisra";

    var con = new Npgsql.NpgsqlConnection(cs);
    con.Open();

    var cmd = new Npgsql.NpgsqlCommand();
    cmd.Connection = con;

    if(dropTableIfExists)
    {
        cmd.CommandText = "DROP TABLE IF EXISTS " +tableName;
        cmd.ExecuteNonQuery();
    }

    cmd.CommandText = "CREATE TABLE " + tableName + "(id SERIAL PRIMARY KEY, status INT, reason VARCHAR(255), ts BigInt)";
    cmd.ExecuteNonQuery();

    con.Close();

    SVApplications.Output("Table " + tableName + " created");

}
```

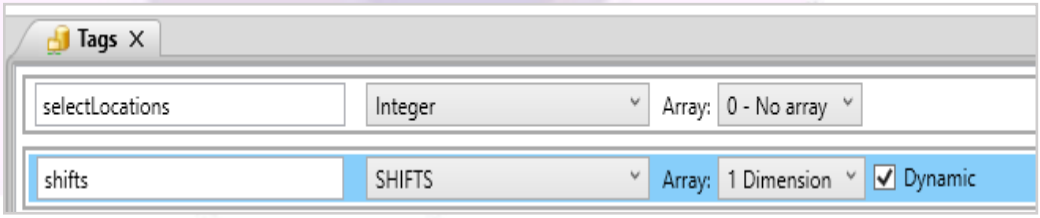
- The function's first parameter is used to create the table name and the second parameter, in case it is true, will drop the table before it is created.

```
28  void InsertEvents(string schemaName, int status, string reason)
29  {
30      string tableName = schemaName + ".events";
31      var cs = "Server=localhost;Port=5433;Username=postgres;Password=Postgres1!;Database=adisra";
32
33      var con = new Npgsql.NpgsqlConnection(cs);
34      con.Open();
35
36      var cmd = new Npgsql.NpgsqlCommand();
37      cmd.Connection = con;
38
39      DateTime currentDate = DateTime.UtcNow;
40      long elapsedTicks = currentDate.Ticks;
41
42      cmd.CommandText = "INSERT INTO " + tableName + "(" + status + ",'" + reason + "'," + elapsedTicks + ")";
43      cmd.ExecuteNonQuery();
44
45      con.Close();
46
47      SVApplications.Output("New value inserted in " + tableName + "table. Reason: " + reason + ", Status: " + status + ", TS: " + elapsedTicks.ToString());
48  }
```

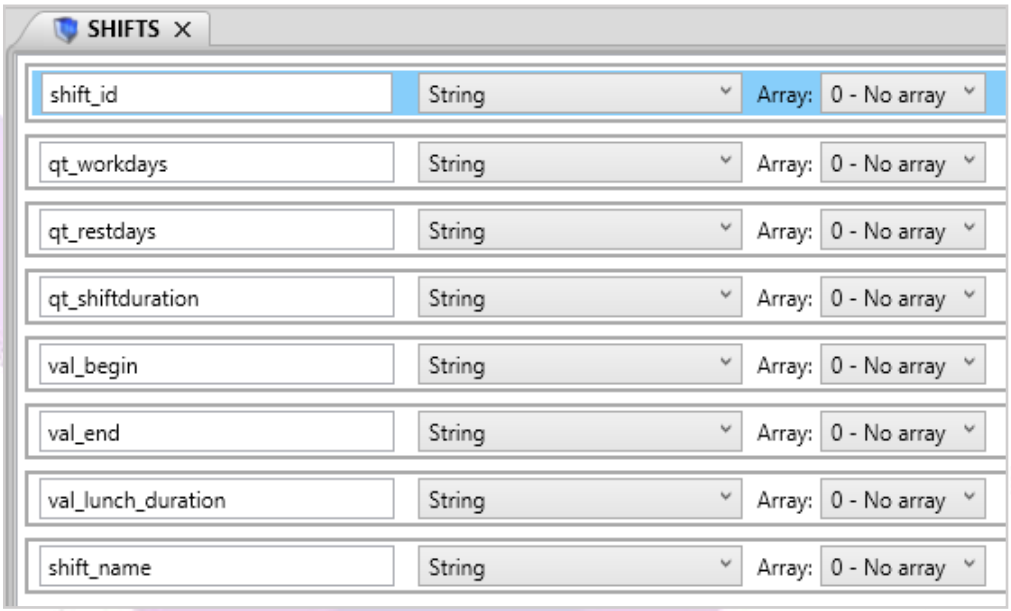
- The function's first parameter sets which schema the function will use, the other two parameters set the value of the “status” and “reason” of the table.

5.2. Using SVDBConnection

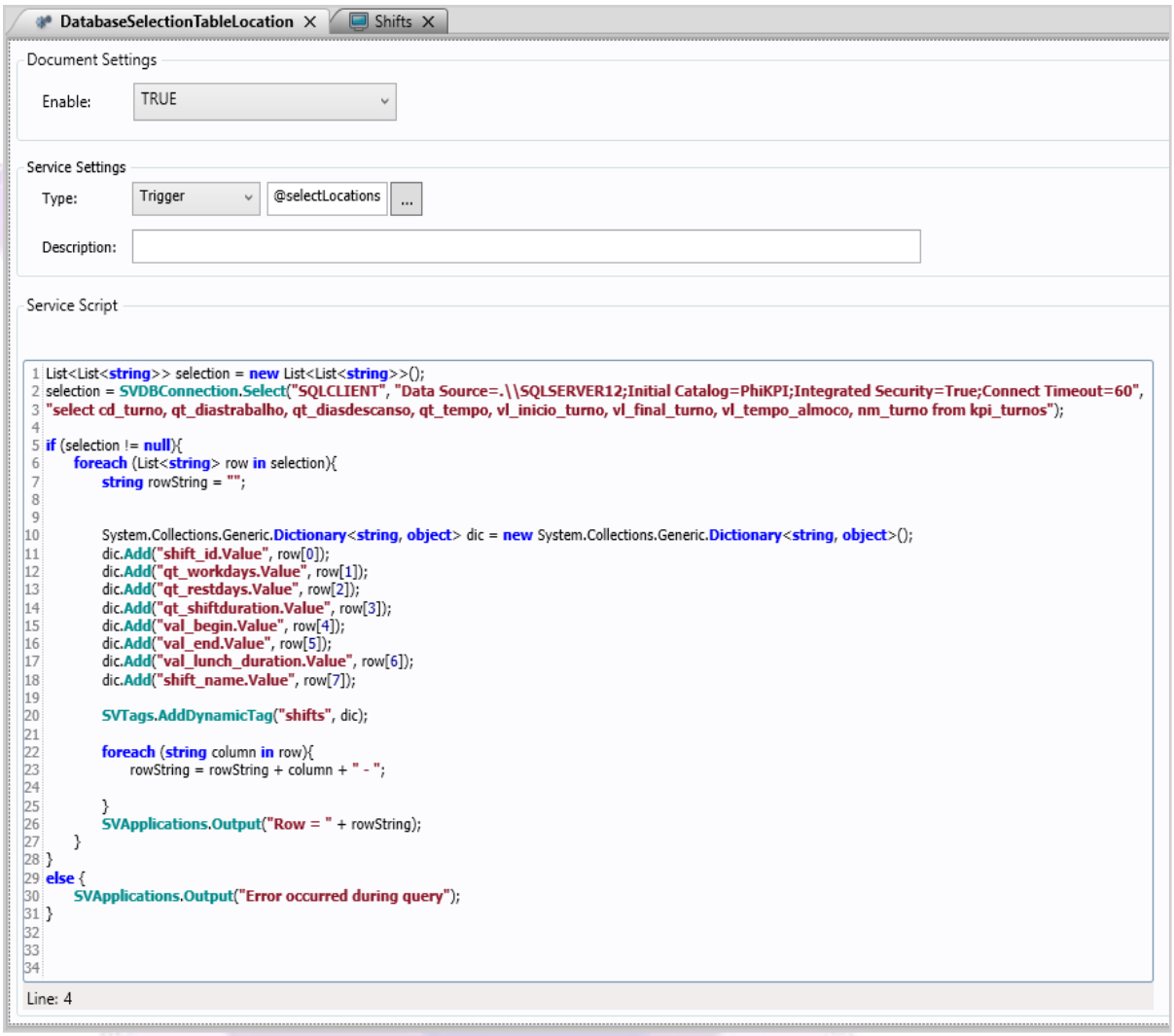
The user may also use the “SVDBConnection” within the System Function Library to run queries. It is like the example in the 4.5 Using SVDBConnection Functions section, but in this example, the user will need to provide the connection string and the provider. The example will show how to select values from a Microsoft SQL Server table and add those values to a dynamic array tag.



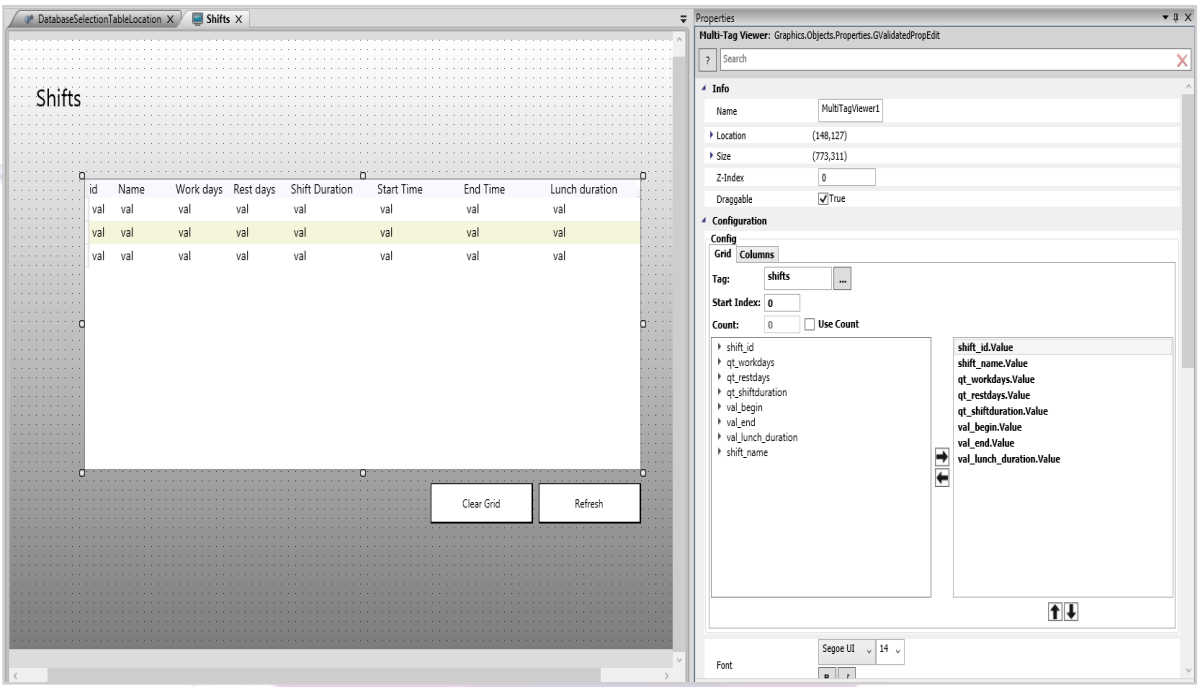
- The SHIFTS data type contains the following tags:



- The following script was inserted in a Service. As it was already detailed, it will connect to a Microsoft SQL Server table, select all the values, and create new entries into the dynamic array tag.



- Create a MultiTagViewer object in a graphic screen and connect it to the dynamic tag with the values as shown in following image.



- Now run the application and check the values inside the object.