ScanMaster Designer Database Marking

1 Introduction

Database Connectivity is useful when it is necessary to retrieve data from external database files such as Microsoft Access® (**.mdb**) and Microsoft Excel® (**.xls**) files. Typical applications for database connectivity include the reading of serial numbers for production unit identification and the reading of barcode information or batch number identification codes. Additionally, geometric information obtained through a database file provides a method of dynamic manipulation of the geometry of the image.

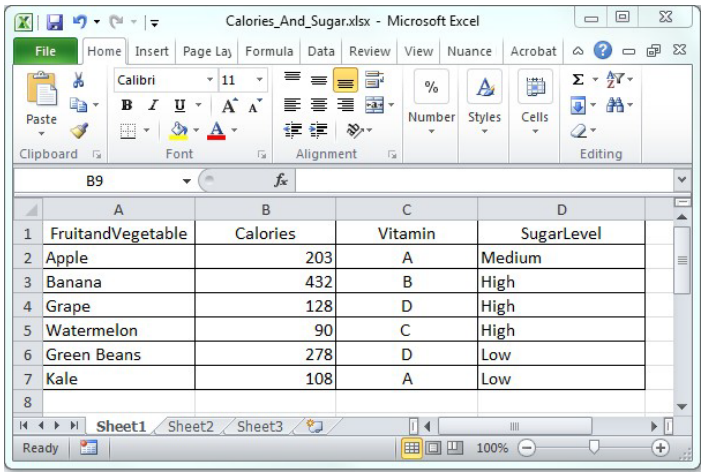
Database Connectivity helps you to automate your repetitious marking, while still maintaining the highest levels of quality.

The following sections are included in this document:

Keywords: Database, Microsoft Access & Excel, ScanMaster Controller, Script, Script Assistant

2 Database and SMD environment preparation

An Excel file named **Calories_And_Sugar.xlsx** will be used in the example presented in this Application Note. This Excel spreadsheet contains different types of fruits and vegetables with their calories, vitamins, and sugar values. The following figure shows **Calories_And_Sugar.xlsx**.



	A	B	C	D
1	FruitandVegetable	Calories	Vitamin	SugarLevel
2	Apple	203	A	Medium
3	Banana	432	B	High
4	Grape	128	D	High
5	Watermelon	90	C	High
6	Green Beans	278	D	Low
7	Kale	108	A	Low

Figure 1 - Example Spreadsheet

3 Database automation creation

Do the following:

1. Left-click the **Database** icon in the **Project | Automation** panel of the ScanMaster Designer

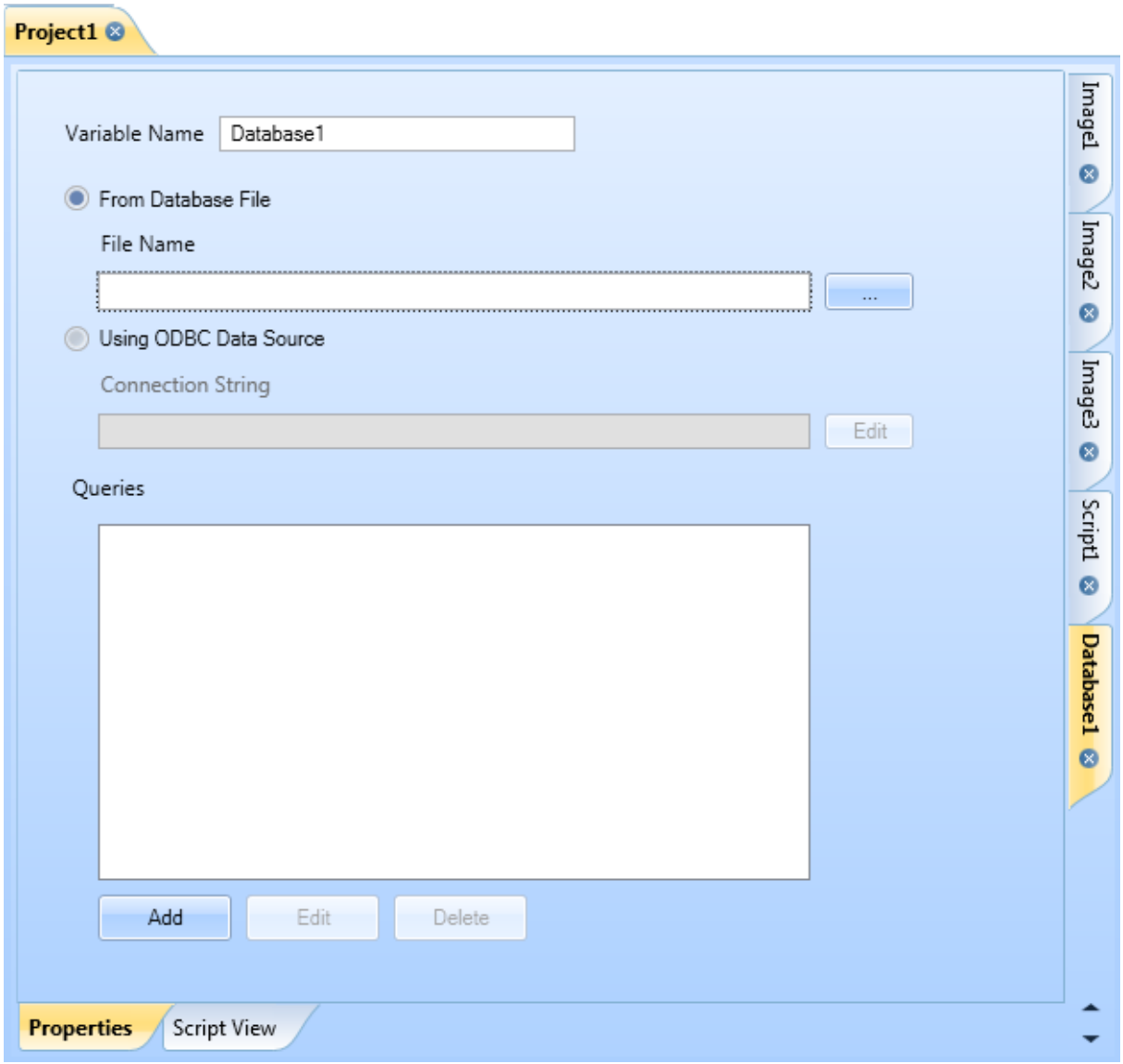


Figure 2 - The Database Connectivity Properties window

2. With the **From Database File** radio button selected, left-click the Browse button ... beside the **File Name** field of the **Database Connectivity Properties** window.

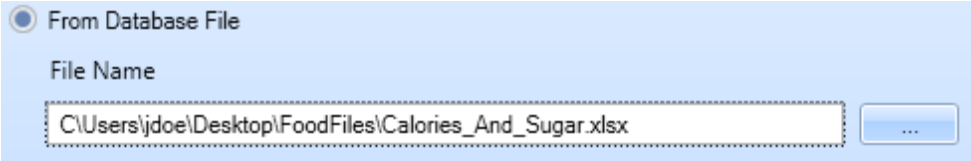


Figure 3 - The File Name field of the Database Connectivity Properties window

4 Add Queries

Queries are used to extract the necessary data from the data source(s) and to implement data filtration. SMD supports simple queries, queries with conditions, and queries with conditions that have dynamic arguments. Refer to the following subsections for examples of how each of the three query types could be implemented for the Excel file **Calories_And_Sugar.xlsx**:

- "Simple Query" (below)
- "Query with a condition" on page 7
- "Query with a condition that has a dynamic argument" on page 9

4.1 Simple Query

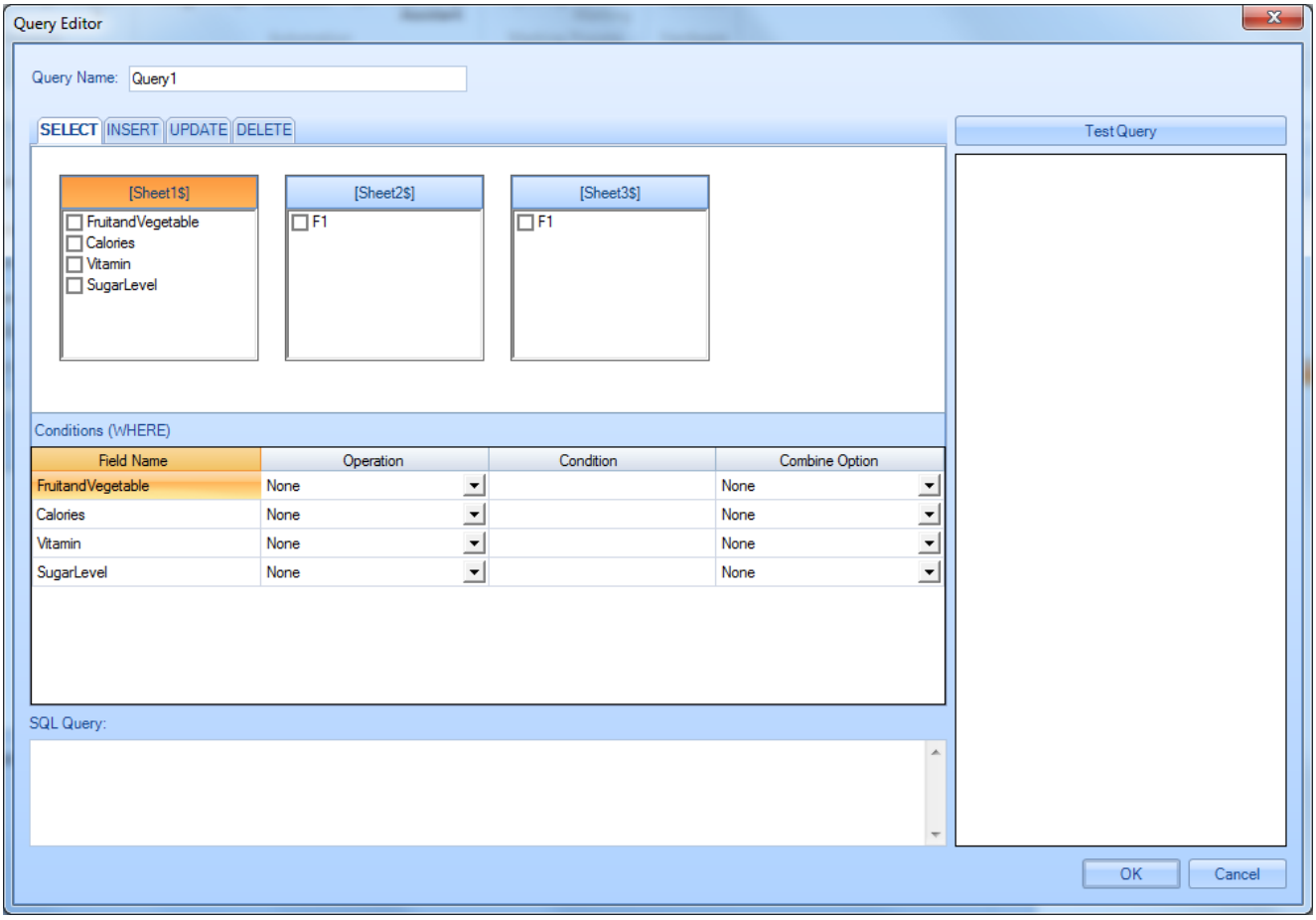
In the following example, a simple query is created to extract the name of the fruit or vegetable and its corresponding calories for marking on a Display plate. The extracted data is then fed dynamically into the marking job.

Do the following to create the simple query:

1. Left-click the Add button in the Database Connectivity Properties window. This displays the Query Editor as shown in the *Figure 4 - The Query Editor* on page 5.
2. Enter a name for the query in the **Query Name** field. The ScanScript code will refer to the query by this name.

   The **Query Editor** includes a table for each worksheet in the Excel file, which are shown as **[Sheet1$]**, **[Sheet2$]** and **[Sheet3$]** in the following figure. **[Sheet1$]** contains the fields in the first worksheet of the Excel file.

   NOTE: As with most Excel worksheets, the first row of the worksheet contains column headers rather than data.



3. Select the **FruitandVegetable** and **Calories** fields in the **[Sheet1$]** table.
4. Press the **TestQuery** button in the upper-right corner of the **Query Editor** to apply the query to the Excel file. The query results (if any) will be shown in the display area beneath the **TestQuery** button (see the following figure).

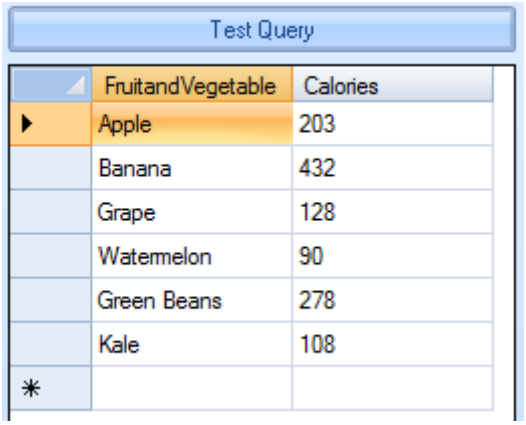


Figure 5 - Query Generated and Tested

5. After confirming that the specified query is working correctly, press the **OK** button in the **Query Editor**. This completes the Database connectivity, and the **Database Connectivity Properties** window will be displayed with the query shown in the **Queries** display area (see the following figure).

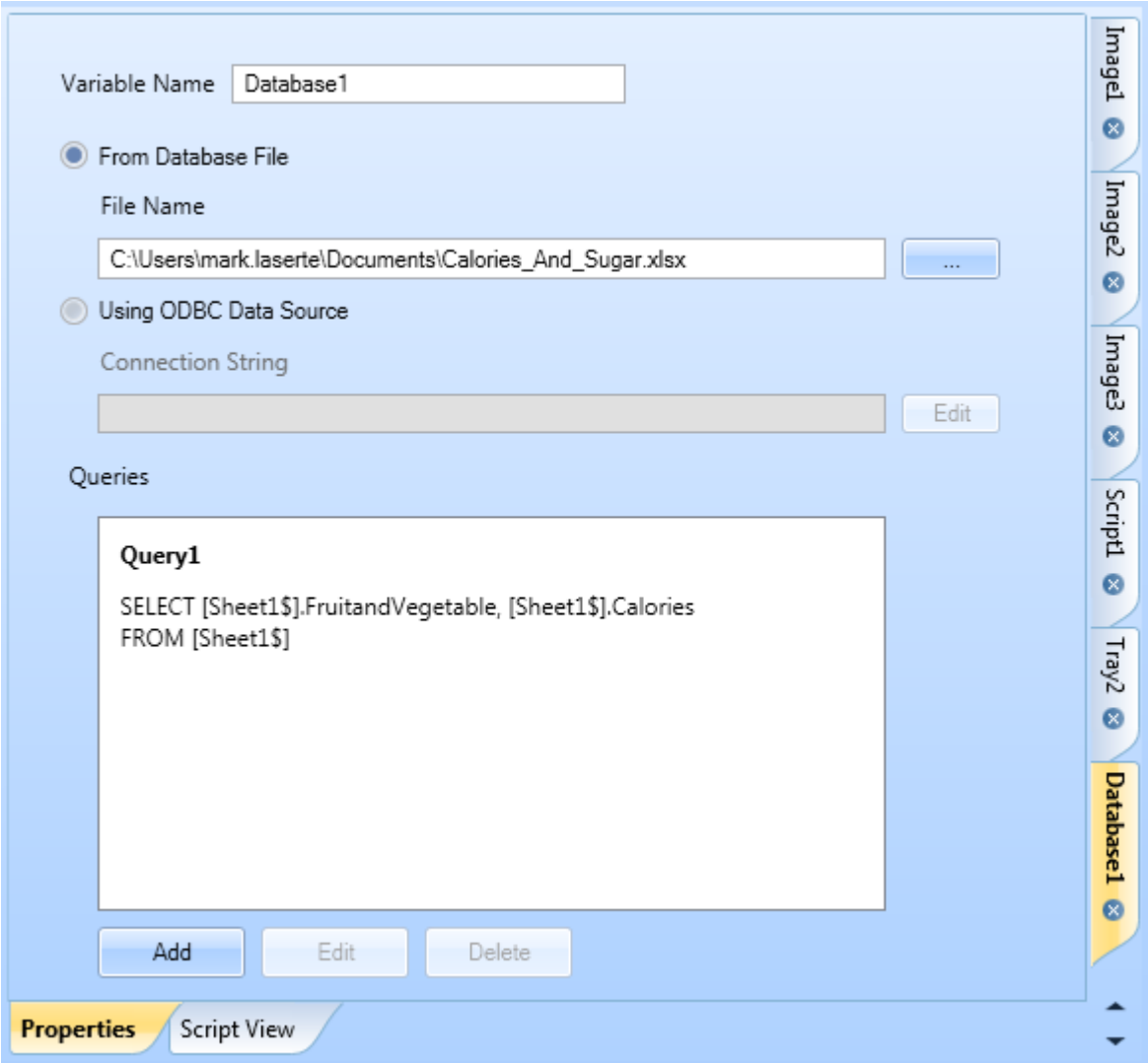


Figure 6 - The Database Connectivity Properties window

Database connectivity generates scripts that might need some adjustments. See Section 5 ("Scripts") on page 10 for more details.

4.2 Query with a condition

Many queries require the data to be fitered by one of column values. In this example, the query will return for marking only those fruits or vegetables that have at least 150 calories.

Do the following to create a query with this condition:

1. Select Query1 in the Queries display area of the Database Connectivity Properties window.

   NOTE: Query1 is the query that was created in Section 4.1 ("Simple Query") on page 4.

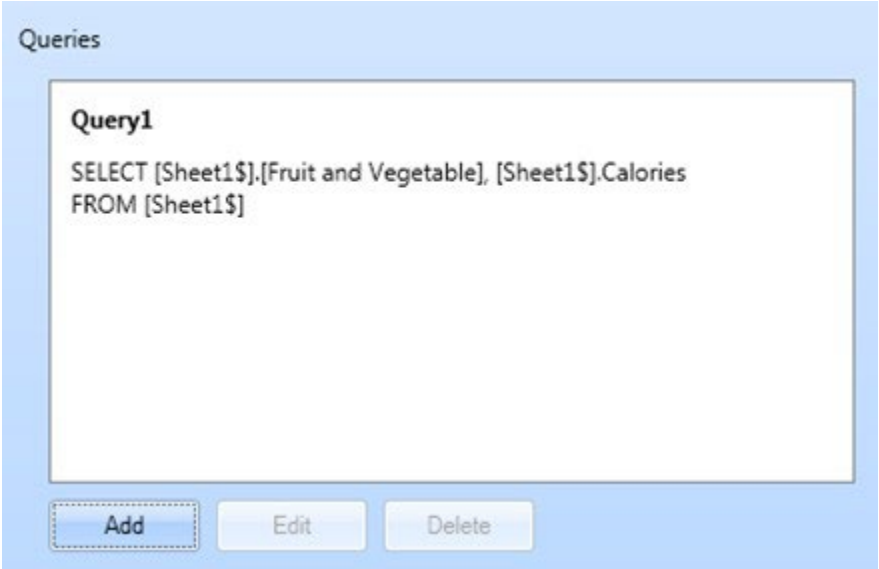


Figure 7 - The Queries display area of the Database Connectivity Properties window

2. Left-click the **Edit** button below the **Queries** display area of the **Database Connectivity Properties** window. This displays the **Edit Query** dialog box as shown in the following figure.

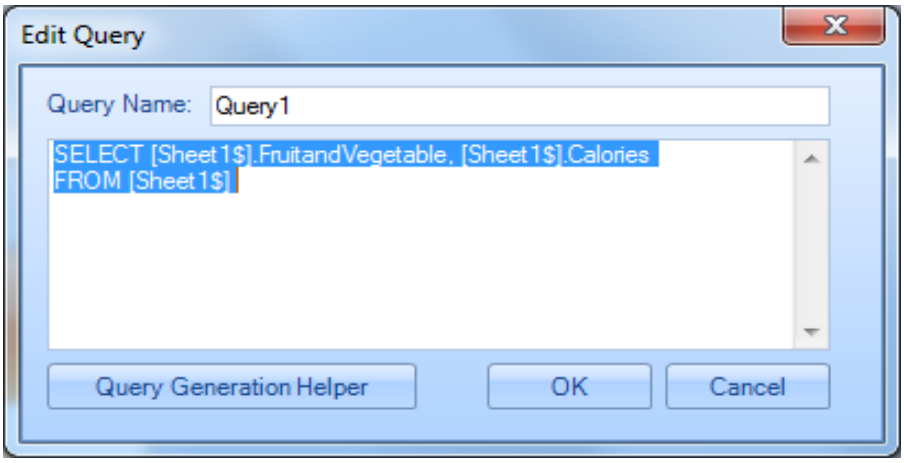


Figure 8 - Edit Query

3. Left-click the **Query Generation Helper** button in the **Edit Query** dialog box. This displays the **Query Editor**.

4. Select **Greater Than** from the dropdown list in the **Operation** column of the **Calories** row.

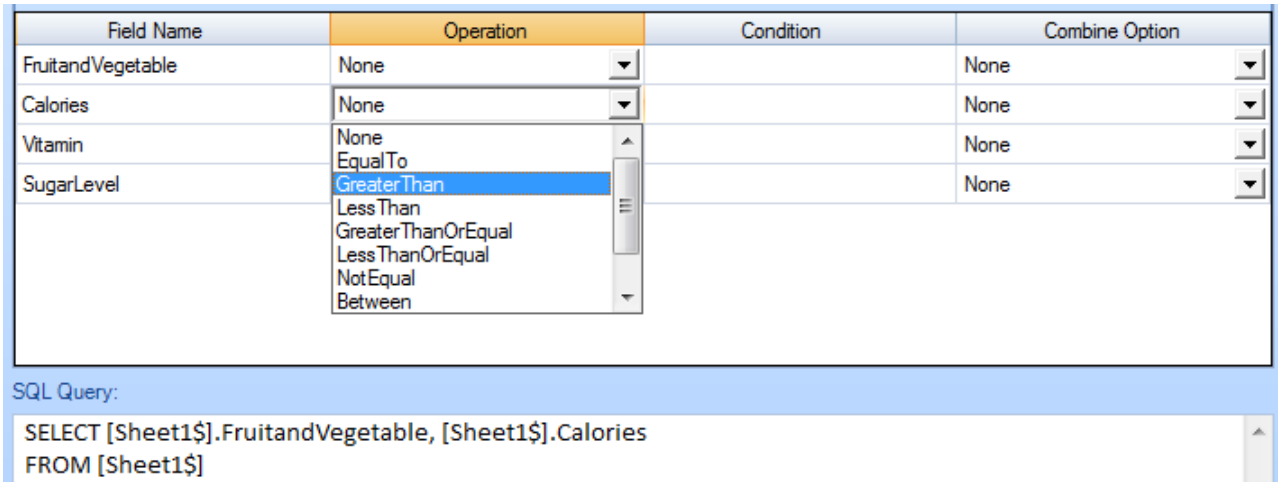


Figure 9 - Query Operation

5. Double-click in the **Condition** field of the **Calories** row, type 150 in that field, and press the **OK** button. The condition will be added as a "WHERE" statement to the query. As shown in the following figure, the updated query will appear in the **Queries** display area of the **Database Connectivity Properties** window.

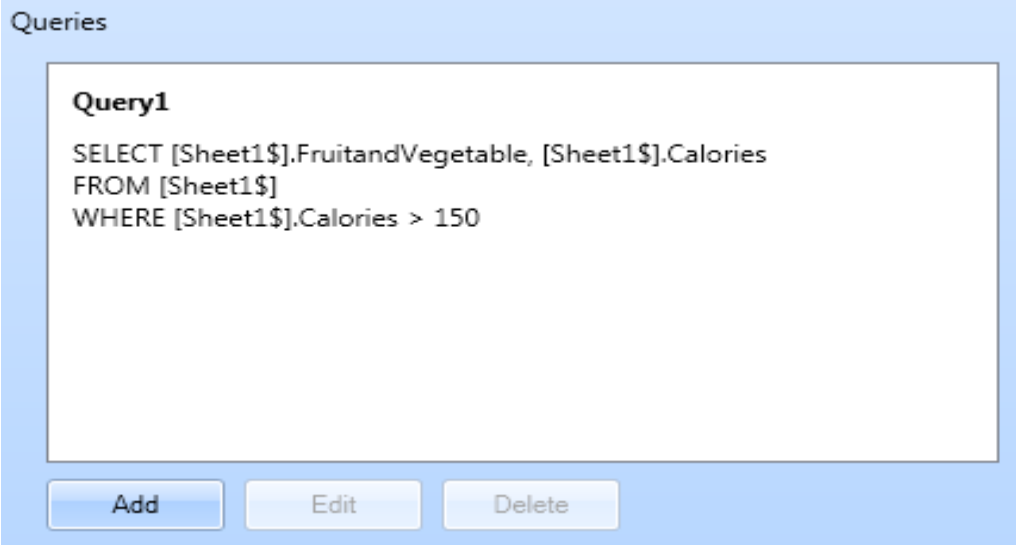


Figure 10 - Updated Query

Because the condition is applied when the database is accessed, the database is already filtered before marking begins. Hence there will not be any change in the script; it will only mark those fruits and vegetables that have a Calories value greater than 150.

Database connectivity generates scripts that might need some adjustments. See Section 5 ("Scripts") on page 10 for more details.

4.3 Query with a condition that has a dynamic argument

The value that you want to use in the **Condition** field of a query may only be known at runtime. When this is the case, you would create a query that has a dynamic argument as its condition. The following example returns for marking those fruits and vegetables whose **Calories** are greater than "x" when the value of "x" is specified at runtime.

Do the following to create a query that fits this scenario:

1. To indicate that the value of the condition is provided dynamically, the user has to enter "{variableName}" in the **Condition** field of the **Query Editor**. Do the following to find **FruitAndVegetable** names with calories greater than “x” (which will be specified during runtime):
   a. Specify a **Greater Than** condition in the **Operation** field of the **Calories** row.
   b. Enter **(x)** in the Condition field of the Calories row.

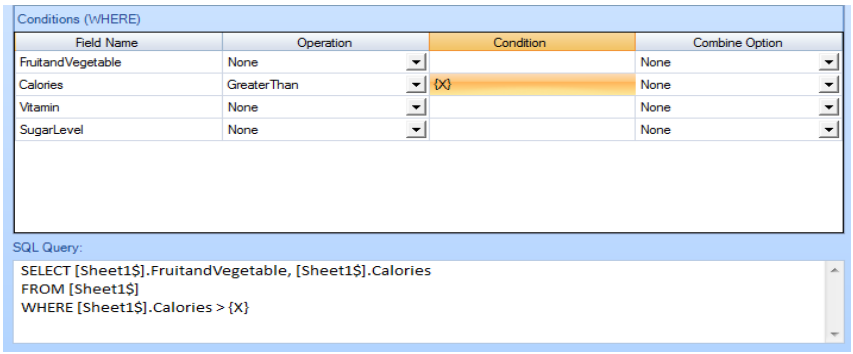


Figure 11 - Dynamic Query Condition

2. As the condition is set with dynamic argument, the script must change to include the value of the argument when defining a query to read a database.

   The following commands dynamically set the **Calories** value to be greater than 200 and select 9 records for marking:

```
Database1.Init() --Initialize database
recordset = Database1.Query1(200) --Get resultant recordset which contains [FruitAndVegetable] and [Calories]
length = recordset.Length --length of the recordset
```

Database connectivity generates scripts that might need some adjustments. See Section 5 ("Scripts") on page 10 for more details.

5 Scripts

Each database generates scripts that will allow the user to see the database variable name and connect the database script with the image script. Click on the **Script View link** in the **Database Connectivity Properties** window to see database script as shown in the following figure.

```
Database1 = {}
Database1_databseTableLocakHandler = {}

--Initialize database
function Database1.Init()
    Database1_database = Database.databaseconnection("Provider=Microsoft.ACE.OLEDB.1
end

local function ExecuteQueryOnDatabase(sqlStatement)
    if (Database1_database ~= nil) then
        local recordset = Database1_database.executequery(sqlStatement)
        return recordset
    else
        error("Database should be initialized, before executing queries")
    end
end

--Query functions
function Database1.Query1()
    local sqlStatment = "SELECT [Sheet1$].FruitandVegetable, [Sheet1$].Calories  FRC

    return ExecuteQueryOnDatabase(sqlStatment)
end



--field validations
function Database1_databseTableLocakHandler.__index(databaseTable, key)
    error("Cannot find "..key)
end

function Database1_databseTableLocakHandler.__newindex(databaseTable, index, value)
    error("Cannot find "..index)
end

setmetatable(Database1, Database1_databseTableLocakHandler)
```

Image1 Image2 Image3 Script1 Tray2 Database1

Properties Script View

Figure 12 - Database query

1. Select the Script Assistant icon in the ScanMaster Desginer Ribbon to view real-time logger thatacts as a server, connecting between the scan card and the dll/database.

NOTE: Always run ScanScript Assistant before you start marking a database.

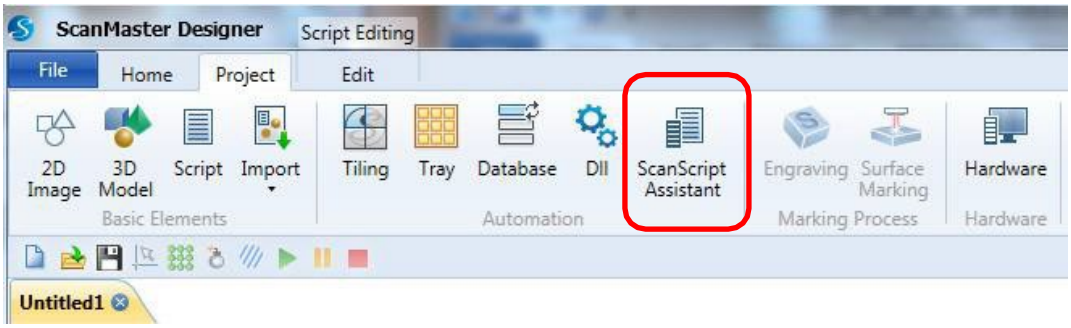


Figure 13 - The ScanScript Assistant icon

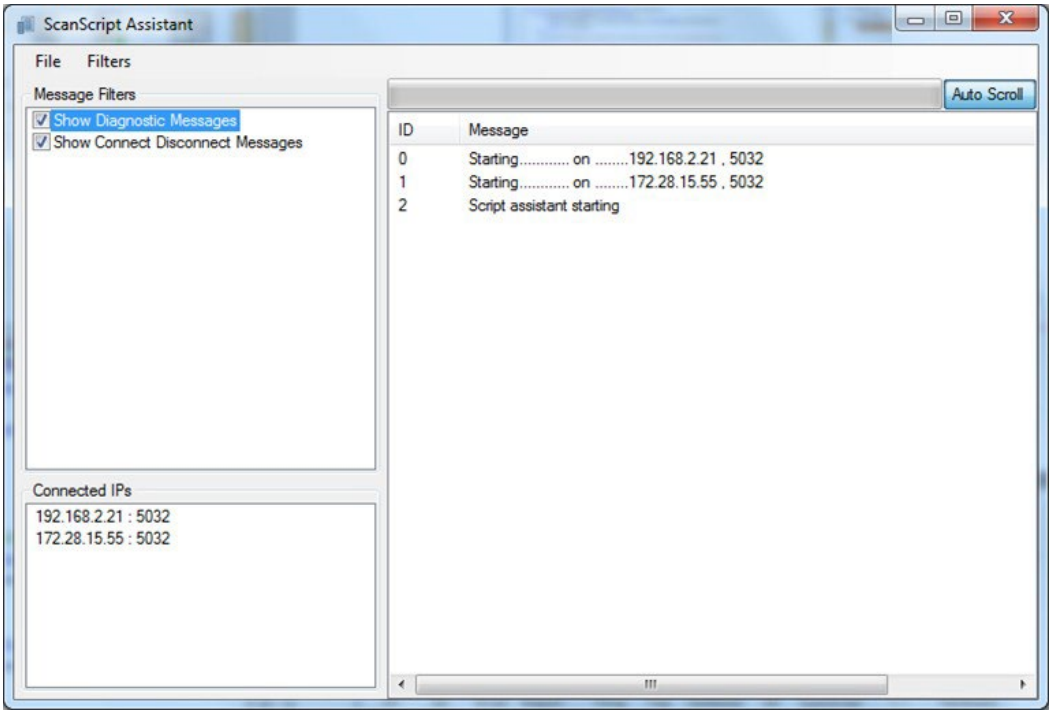


Figure 14 - ScanScript Assistant

1. Writing a script. Database1.Init () has to be called to initialize the database. Here “Database1” is the name of the database automation object.

   To execute the specified query on “Database1”; can be done by calling recordset = Database1.Query1(). Here the recordset is the resultant record set from the query, so a resultant record set only for select queries.
   User can call utility methods and properties of recordset object such as:

Call recordset.Length to get total number of records returned from the query.
Call recordset.ColumnCount to get number of columns in the recordset.
Call recordset.GetColumnNames() to get the column names of the recordset. This will be returned as string array. For loop can be used to iterate through this string array.
Call recordset.GetRecord(recordIndex) to retrieve the specific record from the result. The record object which can be retrieved from recordset.GetRecord(recordIndex) will contain values in each column.
Write recored = recordset.GetRecord(1) to get a value in specific column

And these values can be readily used in any marking purpose.

2. Delete **ScanAll**() from script and complete script example for marking FruitAndVegetable & Calories including marking parameters such as Mark and Jump Speed, all Delays and Laser parameters is as shown in the following script example.

```
SetUnits(Units.Millimeters)
SetAngleUnits(AngleUnits.Degrees)

Laser.Power = 35.0 -- Set the laser power
Laser.Dutycycle1 = 50.0 -- set laser duty cycle
Laser.Frequency = 100.0 -- set laser frequency

Laser.MarkSpeed= 1000.0 -- set the Mark speed
Laser.JumpSpeed = 2000.0 -- set the jump speed

Laser.JumpDelay = 100.0 -- set jump delay
Laser.MarkDelay = 100.0 -- set mark delay
Laser.PolyDelay = 30.0 -- set poly delay
Laser.LaserOnDelay = -100 -- set Laser ON delay
Laser.LaserOffDelay = 120 -- set Laser OFF delay
-----------------------------------

--Connect with script assistant
ScriptAssistant("192.168.2.21", 5032)

--Initialize database
Database1.Init()

--Get resultant recordset which contains [FruitAndGegetable] and
[Calories]
recordset = Database1.Query1()

--length of the recordset
length = recordset.Length
Report("Length of recordset "..length)
--Number of columns in the returned recordset
columnCount = recordset.ColumnCount
Report("Number of columns "..columnCount)
--get column names this will be a string array
columnNames = recordset.GetColumnNames()
--iterate though column Names array and get column names
for columnIndex=1,columnCount do
    Report(columnNames[columnIndex])
end

--iterate through records
```

```
for recordIndex=1,length do

    record = recordset.GetRecord(recordIndex)
    local FruitAndVegetable = record[columnNames[1]]
    local Calories = record[columnNames[2]]
    Report(FruitAndVegetable.." "..Calories.." ") -- Reporting to
Output message window

    FruitText = Text.Horizontal()
    FruitText.Font = "SIMPLEX.OVF"
    FruitText.X =-40
    FruitText.Y =0
    FruitText.Height =2
    FruitText.text = record[columnNames[1]]
    Image.Text(FruitText) --Mark FruitAndVegetable

    FruitText.X = -20
    FruitText.text = record[columnNames[2]]
    Image.Text(FruitText) -- Mark Calories

    Laser.WaitForEnd()
    message = Smd.MessageBox("Marked Record #"..recordIndex.. " Press
Yes to mark Next Record", "Next Record", Smd.MessageBoxButton.YesNo,
Smd.MessageBoxIcon.Question)

    if (message == "Yes") then
            Report("Marked Record #"..recordIndex)
    else
        break
    end
end
```

The above script executes the query specified in database automation editor and it will mark FruitAndVegetable and Calories in same line. It will populate the Message Window as shown below showing that record # is marked and system is ready to mark for next record as seen in ***Figure 15 - Record Message*** (below)

Figure 15 - Record Message