MS-SQL Integration

# In Unreal Engine 4

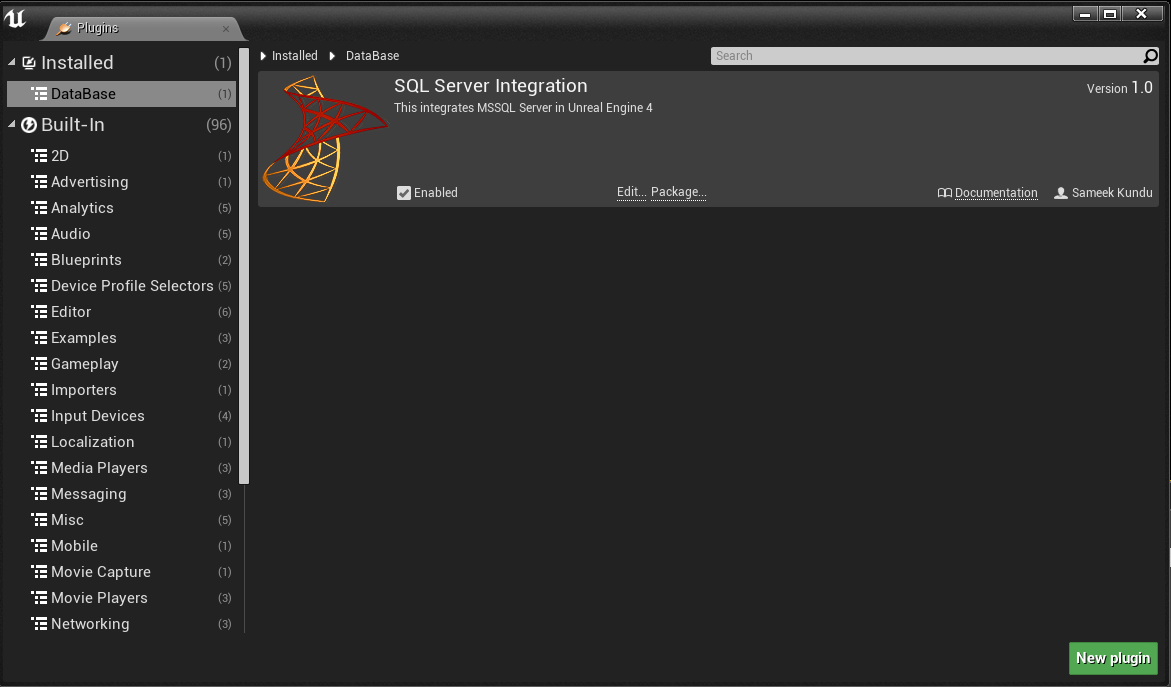
# User Manual

MS-SQL Integration is a plugin that lets you connect to your SQL Server database, and store and retrieve data from the server via SQL queries, directly from the Blueprint. This plugin brings the power of C#.NET in Unreal Engine 4.

If you have worked with MS SQL Server, you must be very familiar to Select, Update, Insert and Delete SQL queries, joins and sub-queries. If you know C#.Net, and worked with ADO.Net, you must be knowing how to set Connection properties for your SQL Server, and how to write and execute your SQL Queries within your .NET application.

But now you can use Blueprints in Unreal Engine 4 to do your job. You are still writing the same queries that you are familiar with, but here you are writing it within Blueprints, which gives you flexibility to store and retrieve data as well as Images from your SQL Server to your UE4 game.

To get started, first ensure that the plugin is installed, by visiting the Plugin Window, as shown below:



If you are not familiar with SQL Server at all, you will get plenty of online resources to get you started, and you can definitely test it out by downloading the free version, SQL Server Express.

SQL Server Express is an entry level database server having a limitation of 10 GB size, which is perfect from small scale applications.

Here I will be explaining you in details how to set up your database connection and how to write and execute SQL queries from UE4.

*Working with SQL Server Express*

**If you already have your database and tables ready to be integrated with your UE4 project, you may skip this section.**

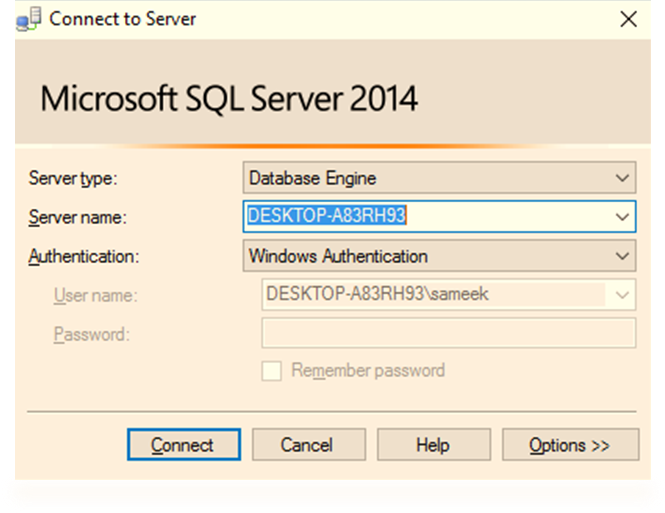
Assuming that you have your database server up and running, either locally or remotely, to get started working with this plugin, you first need to create a database and a table in your server, where you would like to store all your in-game data.

If you are unsure regarding how to work with SQL Express, let me quickly walk you through the basic steps.

1> Download and install SQL Server Express edition from the below link:

<https://go.microsoft.com/fwlink/?LinkID=799012>

2> Open your SQL Server client and connect to your database server by using Windows Authentication or SQL Server authentication.



By default, you will get a local DB server called **locahost**, which will be saved in your local machine. To create a dedicated server, please go through the steps given in the below link:

<https://docs.microsoft.com/en-us/azure/sql-database/sql-database-get-started>

To understand how to create a login, please visit the below link:

<https://msdn.microsoft.com/en-in/library/aa337562.aspx>

3>Once you are connected to the server, you need to create a new database. Visit the below link to help you out with the database creation:

<https://msdn.microsoft.com/en-us/library/ms186312.aspx>

 4>Once your database is created, it should be visible in object explorer, within **Databases.**

Right click on your database name, and open New Query Window. Then create one or more tables, which you want to use for storing and retrieving your game data. Below is a simple query that lets you create a table called **Products**. (Source - <https://msdn.microsoft.com/en-us/library/ms365315.aspx>)

CREATE TABLE dbo.Products

(

ProductID int PRIMARY KEY NOT NULL,

ProductName varchar(25) NOT NULL,

Price money NULL,

ProductDescription text NULL

)

GO

*Setting Up Database Connection in UE4*

Before we begin writing our SQL queries and executing them in our database, we first need to set up a connection within UE4. This is a onetime setup, and is similar to how we manually connect to our DB Server using Windows or SQL Server Authentication, as shown in the above steps.

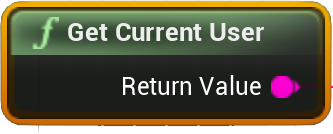
Authentication Modes

To set up a DB Connection, we either need to use Windows Authentication mode or SQL Authentication mode.

**Windows Authentication mode** authenticates users to use the Database server via Windows Login ID and Password. The following functions are needed to setup a Windows Authentication.

**Get Current User –**

This function returns the current Windows User Name.



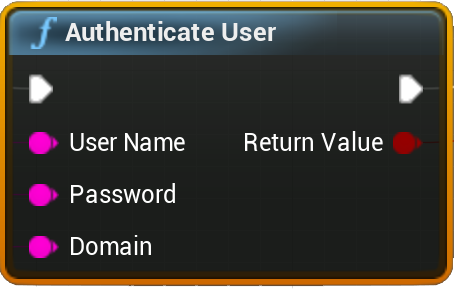
**Get Current Domain –**

This function returns the current PC Domain



**Authenticate User –**

This function takes the current user name, domain and password for your Windows account as input and returns the authentication result as a Boolean variable. Please note that this function will not work if the current Windows user does not require any password to login. So you may want to avoid using this function unless absolutely necessary.



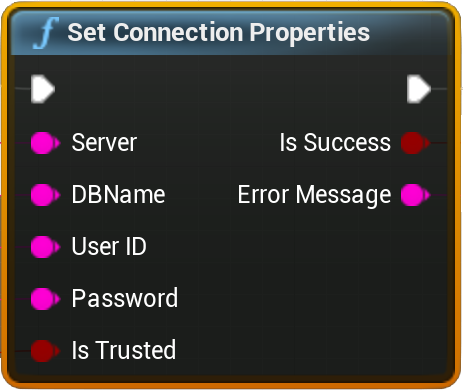
**SQL Server Authentication mode** uses logins that are created and stored in SQL Server and are not based on Windows user accounts. Here you need to explicitly enter both your SQL Server ID and password, while setting up the connection property.

To learn about Authentication modes in detail, you can refer to the below link:

<https://msdn.microsoft.com/en-us/library/ms144284.aspx>

 SQL SERVER CONNECTION

To connect to the SQL Server, we first need to set the connection properties. For that, we need to call the function **Set Connection Properties**, which forms a valid connection string that will later be used to access the given database. If you are using Windows Authentication, you don’t need to enter **Used ID** or **Password**.



**Inputs**

**Server** –

Here you need to enter you Server address. If the server is your local machine, then it is ***localhost*** by default. If you have hosted your server in your web service or using the steps described [here](https://docs.microsoft.com/en-us/azure/sql-database/sql-database-get-started), you must be having your dedicated server name. You need to enter that here.

**DBName** –

This is your Database name where you want to set up your initial connection.

**User ID** –

For SQL Server Authentication, you need to pass your SQL Server user name here.

**Password** –

For SQL Server Authentication, this is your SQL Server password.

**Is Trusted** –

A Windows Authentication is referred to as a trusted connection to the database, since any user can login from their Windows user accounts. This should be **checked** if you are using Windows Authentication mode.

**Outputs**

**Is Success** –

This will return true if the database connection has been successfully established, else false.

**Error Message** –

This will be empty if connection to the database is successful, else it will contain the exception message.

*Writing SQL queries within UE4 Blueprints*

Once the connection property has been set, you can now start forming your SQL queries. Forming queries can be categorized in two types, queries that modify data to the database (INSERT, UPDATE, DELETE), or the queries that fetch data from the database (SELECT). We will discuss these two categories one by one.

Let us say we have a created a table in our database called **Products** by using the below query:

CREATE TABLE dbo.Products

(

ProductID int PRIMARY KEY NOT NULL,

ProductName varchar(25) NOT NULL,

Price money NULL,

ProductDescription text NULL

)

We will take this table as a reference to form our queries and discuss the two categories of query formation.

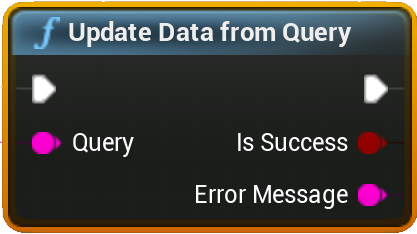
Please note, the functions that you need to perform query execution in your server will usually have two common outputs:

**Is Success** –This will return false if the query execution fails

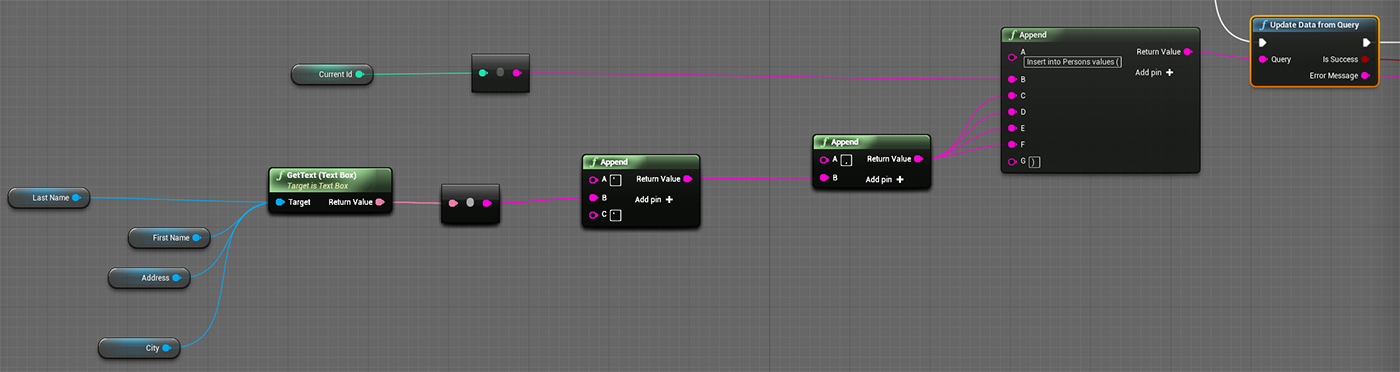
**Error Message** –This will have the exception message that will be thrown at the SQL server end. If the query execution is successful, this will be an empty string.

 UPDATE QUERIES

To modify data in our table **Products**, we will use the function **Update data from Query**, which takes a string input where we have to enter our query and executes the query in the database.

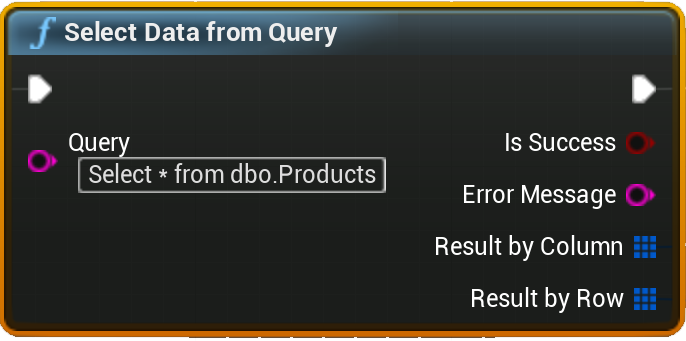


A typical way of forming a simple Insert query is shown below.



 SELECT QUERIES

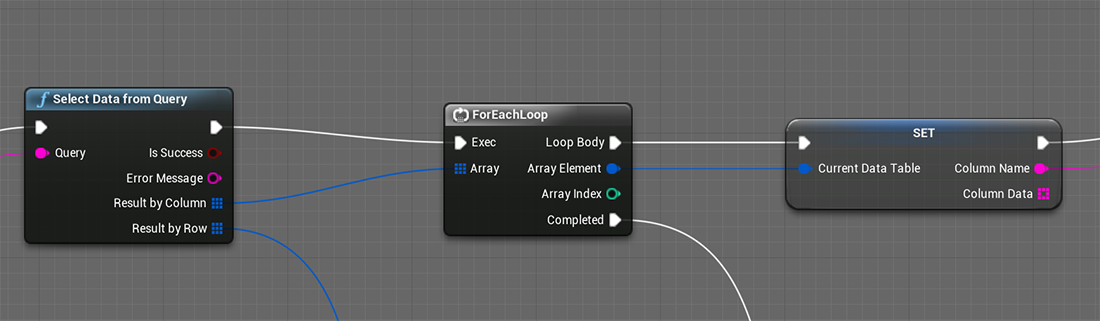
To bring the data from SQL Server to your game, you need to call the function **Select Data from Query**. This will take your select query as a string input.



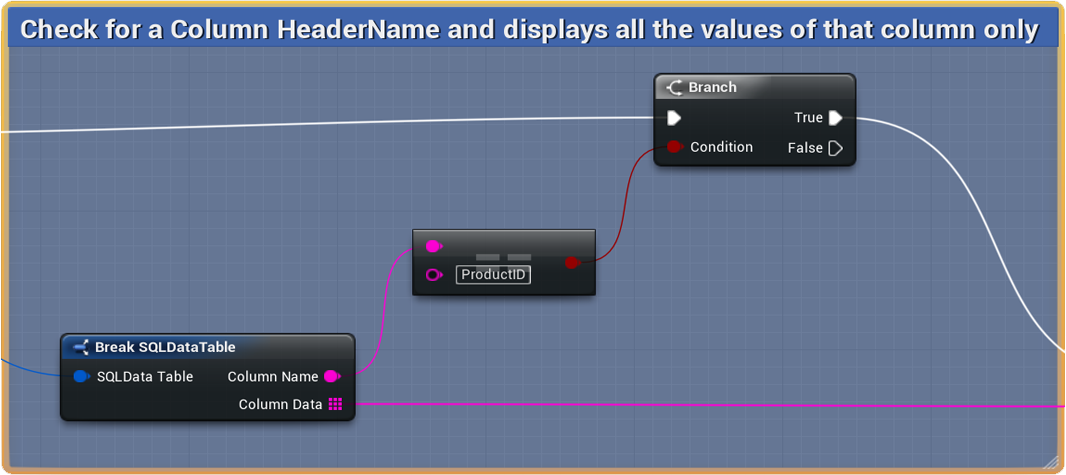
**Outputs**

**Result by Column** –

This will return an array of structured variable called **SQLDataTable**. This represents a column in the table and contains two parameters, **ColumnName** and **ColumnData**.



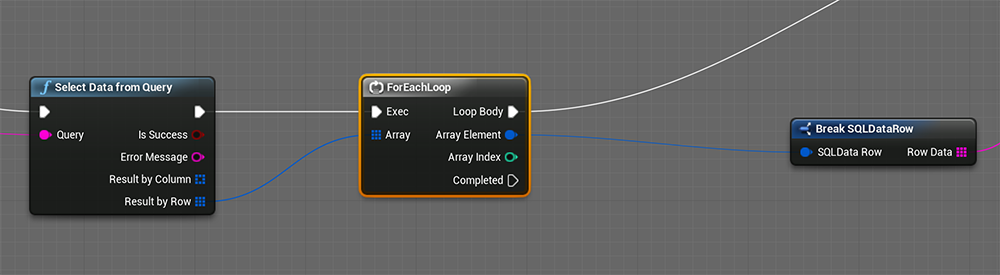
**ColumnName** contains the header name of the column of the current instance of our initial array that we get by executing our select query. If you want to display data of a specific column, you can use this variable to match the column name, as shown below.



**ColumnData** is an array of string consisting of all the elements of a given column. So you can easily set up a for-loop here to get all the elements of a particular column iteratively.

**Result by Row—**

This contains an array of structured variable called **SQLDataRow**. This structure contains an array of string that represents values of an entire row. So the number of elements in this string array equals number of columns, and number of elements in the **Result by Row** array equals number of rows in the output of the Select query.



***Please note that there is no restriction of using select query with this function. Any Select query working within SQL client will work here just fine.***

*Storing and Retrieving Images from SQL Server*

 If you have expertise in SQL Server, chances are you are very much familiar with working with **ImageType** data that helps in storing and retrieving images from SQL Server. In this section, we talk about how we can work with **ImageType** in UE4.

To work with images, we first need to create one table in the database that has an **ImageType** datatype. Let us assume we have created such a table, by executing the below query.

CREATE TABLE [dbo].[ImageData] (

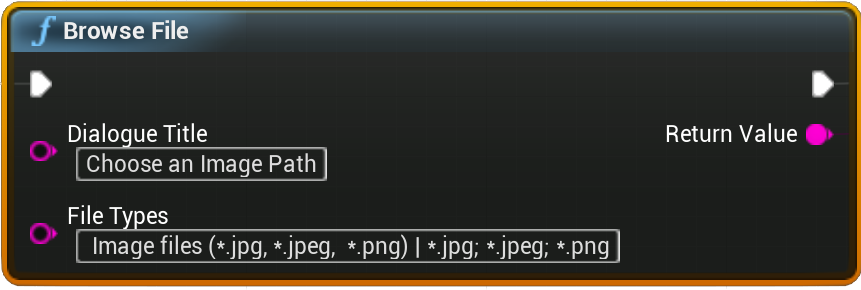
[ID] [int] IDENTITY (1,1) NOT NULL ,

[Picture] [image] NULL

) ON [PRIMARY] TEXTIMAGE\_ON [PRIMARY]

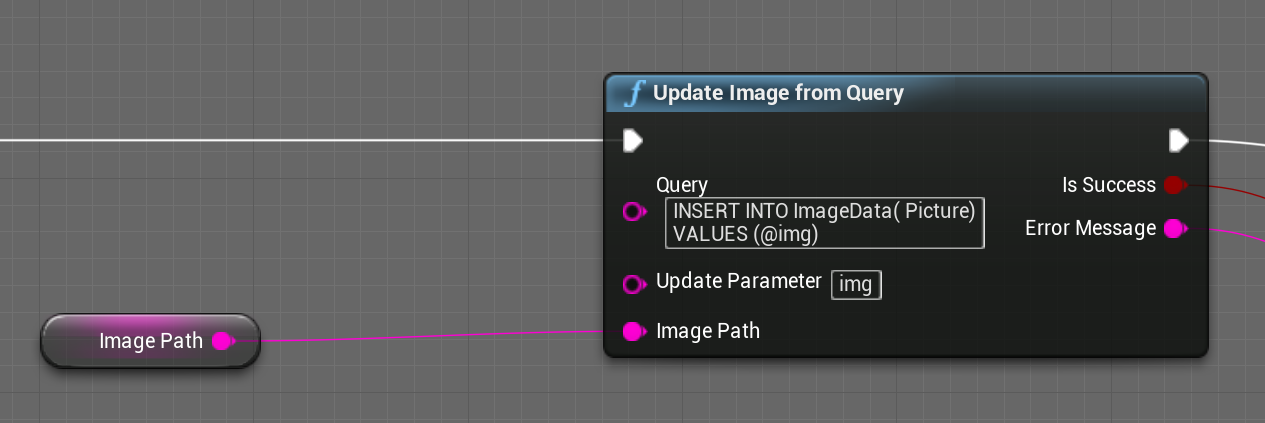
 Update Images

To save an image to our table, we first need to browse for an image from our desktop. As of now, you cannot directly save UE4 Textures to your database. Instead, you can directly browse for an image of common types via Blueprints, by calling the function called **Browse File.**

****

This function will bring up a File Browser dialogue. On selecting the image file and clicking OK, the dialogue box will close and the function returns the file path. You may have your separate ways of retrieving image path from your hard drive, so this function is pretty much optional.

Once we have our image file path, we need to fire the function called **Update Image from Query** that will update the image to the server.



**Inputs**

**Query** – Here we enter our query. So if we want to insert an image in our Picture column, we need to enter it like this:

INSERT INTO ImageData( Picture)

VALUES (@img)

If we want to insert values in other columns, we can do that as well.

INSERT INTO ImageData( ID , Picture)

VALUES (0, @img)

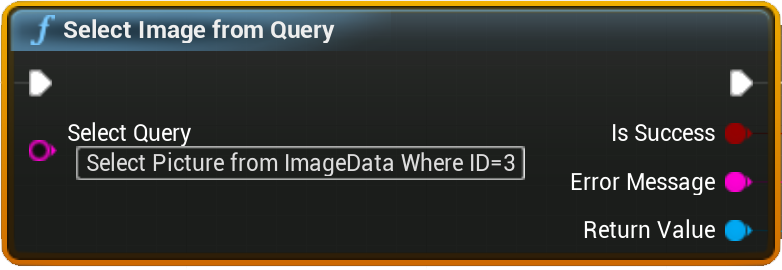
**Update Parameter** –This will contain the variable name which should be replaced by our image. In the query above, we are using the word **@img**. This is a common way of inserting dynamic data in SQL server, and you must be familiar with it if you have hands on experience in ADO.NET. So this input will take the same parameter as input, without the **@** symbol, to help the program determine what variable is being used to represent image variable. So here we need to enter **img**, as shown in the picture above.

**Image Path**—Here we are providing the image file path that we have received from the previous section.

SELECT Images

To select an image from SQL Server, we simply need to call the function called **Select Image from Query** and enter our selection query here.

You need to remember that you can select only one image at a time, so make sure your selection query only fetches a single image and nothing else. For example, when your table has a multiple rows, you cannot write **Select Picture from ImageData** as that will fetch multiple values. Instead, you need to write something like **Select Picture from ImageData Where ID=3**.



This function returns a Texture2D format, which you can use to display the image in your game.

