**Receive Notifications when a Table Record Changes with C#**

**By**

****

[**Christian Del Bianco**](https://dzone.com/users/2607609/christiandelbianco.html)

**·**

**Sep. 16, 15 · Tutorial**

**Like**[**(5)**](https://dzone.com/articles/receive-notifications-with-new-values-when-table-r)

**Comment (2)**

**Save**

[**Tweet**](https://dzone.com/articles/receive-notifications-with-new-values-when-table-r)

[**Share**](https://www.linkedin.com/sharing/share-offsite/?url=https://dzone.com/articles/receive-notifications-with-new-values-when-table-r)

**139.7K Views**

Join the DZone community and get the full member experience.

[**Join For Free**](https://dzone.com/static/registration.html)

The SQLDependency object represents a query notification dependency between an application and an instance of SQL Server. An application can create a SqlDependency object and register to receive notifications via the OnChangeEventHandler event handler every time a record is changed.

However, what it cannot do, is returning the values for the inserted, modified or deleted record.

So, supposing our application has a cache containing record monitored from SqlDependency, for every notification, in order to refresh the cache we have to execute again our select.

**SqlTableDependency**

[SqlTableDependency](https://github.com/christiandelbianco/monitor-table-change-with-sqltabledependency) is a generic C# component used to receive events containing the values for the modified, inserted or delete record. So, supposing our application has a cache filled with data got from a specific table, when its content change, we will receive an event containing a C# object with all property filled with column table values.

**How it Works**

When instantiated, [SqlTableDependency](https://github.com/christiandelbianco/monitor-table-change-with-sqltabledependency" \t "_blank) create for us, a series of database objects, used to monitor a predefined table. The main object are:

1. Queue: containg a message with the modified record values.
2. Service Broker: insert the message in the queue.
3. Table Trigger: for each insert, update or delete operation, prepare the an XML message and use the Service Broker to insert it in the queue.

**Watch Dog Time Out**

[SqlTableDependency](https://github.com/christiandelbianco/monitor-table-change-with-sqltabledependency) implement the IDisposable interface, in order to remove the database object created. Infact it is a good practice is to wrap SqlTableDependency within a using statement or, alternatively, call the Stop() method once we do not need any more notifications. So, when the application will **not**disconnect abruptly, this approach is enough to remove the SqlTableDependency infrastructure (Trigger, Service Broker service, Queue).  
  
However, when the application exits abruptly – that is not calling the Stop() method or not implementing the using statement - we need a way for cleaning up the SqlTableDependency infrastructure. The Start() method, has **watchDogTimeOut**optional parameter used to remove all the database objects. Its default value is 180 seconds: after this amount of time, if there are no listeners waiting for notifications, the SqlTableDependency infrastructure will be removed.

**How to Use**

In this example I created a very simple WPF application with a grid, simulating some stocks value.

Let's start assuming the following database table:

1

CREATE TABLE [dbo].[Stocks](

2

[Code] [nvarchar](50) NULL,

3

[Name] [nvarchar](50) NULL,

4

[Price] [decimal](18, 0) NULL)

5

GO

6

INSERT [dbo].[Stocks] ([Code], [Name], [Price]) VALUES (N'MCD', N'McDonald Corp', CAST(333 AS Decimal(18, 0)))

7

INSERT [dbo].[Stocks] ([Code], [Name], [Price]) VALUES (N'NKE', N'Nike Inc', CAST(240 AS Decimal(18, 0)))

8

INSERT [dbo].[Stocks] ([Code], [Name], [Price]) VALUES (N'DIS', N'Walt Disney Co', CAST(130 AS Decimal(18, 0)))

9

INSERT [dbo].[Stocks] ([Code], [Name], [Price]) VALUES (N'UTX', N'United Technologies Corp', CAST(130 AS Decimal(18, 0)))

10

INSERT [dbo].[Stocks] ([Code], [Name], [Price]) VALUES (N'MSFT', N'Microsoft Corp', CAST(130 AS Decimal(18, 0)))

11

INSERT [dbo].[Stocks] ([Code], [Name], [Price]) VALUES (N'PFE', N'Pfizer Inc', CAST(130 AS Decimal(18, 0)))

12

INSERT [dbo].[Stocks] ([Code], [Name], [Price]) VALUES (N'INTC', N'Intel Corp', CAST(130 AS Decimal(18, 0)))

13

INSERT [dbo].[Stocks] ([Code], [Name], [Price]) VALUES (N'KO', N'Coca Cola Co', CAST(130 AS Decimal(18, 0)))

14

GO

This table is constantly updated with new stock values. We want keep update our grid without use of any polling system. So, what we need is a notification coming from the database every time a record in the table has been changed.

We install the SqlTableDependency [package](https://www.nuget.org/packages/SqlTableDependency/) from visual studio using the following command:

|  |
| --- |
| PM> Install-Package SqlTableDependency |

Now we define a C# model mapping the interested table columns.

1

public class Stock

2

{

3

public decimal Price { get; set; }

4

public string Symbol { get; set; }

5

public string Name { get; set; }

6

}

As you can see, model's properties name can have a different table column name. In this case, we need a mapper to correclty bind the values:

1

var mapper = new ModelToTableMapper<Stock>();

2

mapper.AddMapping(model => model.Symbol, "Code");

Finally we create our SqlTableDependency:

1

\_dependency = new SqlTableDependency<Stock>(\_connectionString, "Stocks", mapper);

2

\_dependency.OnChanged += \_dependency\_OnChanged;

3

\_dependency.OnError += \_dependency\_OnError;

4

\_dependency.Start();

Below the complete code:

1

public partial class Window1 : Window

2

{

3

private IList<Stock> \_stocks;

4

private readonly string \_connectionString = "data source=.;initial catalog=TableDependencyDB;integrated security=True";

5

private readonly SqlTableDependency<Stock> \_dependency;

6

public Window1()

7

{

8

this.InitializeComponent();

9

this.McDataGrid.ItemsSource = LoadCollectionData();

10

this.Closing += Window1\_Closing;

11

​

12

var mapper = new ModelToTableMapper<Stock>();

13

mapper.AddMapping(model => model.Symbol, "Code");

14

​

15

\_dependency = new SqlTableDependency<Stock>(\_connectionString, "Stocks", mapper);

16

\_dependency.OnChanged += \_dependency\_OnChanged;

17

\_dependency.OnError += \_dependency\_OnError;

18

\_dependency.Start();

19

}

20

​

21

private void Window1\_Closing(object sender, System.ComponentModel.CancelEventArgs e)

22

{

23

\_dependency.Stop();

24

}

25

​

26

private void \_dependency\_OnError(object sender, TableDependency.EventArgs.ErrorEventArgs e)

27

{

28

throw e.Error;

29

}

30

​

31

private void \_dependency\_OnChanged(object sender, TableDependency.EventArgs.RecordChangedEventArgs<Stock> e)

32

{

33

if (\_stocks != null)

34

{

35

if (e.ChangeType != ChangeType.None)

36

{

37

switch (e.ChangeType)

38

{

39

case ChangeType.Delete:

40

\_stocks.Remove(\_stocks.FirstOrDefault(c => c.Symbol == e.Entity.Symbol));

41

break;

42

case ChangeType.Insert:

43

\_stocks.Add(e.Entity);

44

break;

45

case ChangeType.Update:

46

var customerIndex = \_stocks.IndexOf(\_stocks.FirstOrDefault(c => c.Symbol == e.Entity.Symbol));

47

if (customerIndex >= 0) \_stocks[customerIndex] = e.Entity;

48

break;

49

}

50

​

51

this.McDataGrid.Dispatcher.Invoke(DispatcherPriority.Background, new Action(() =>

52

{

53

this.McDataGrid.Items.Refresh();

54

}));

55

}

56

}

57

}

58

​

59

private IEnumerable<Stock> LoadCollectionData()

60

{

61

\_stocks = new List<Stock>();

62

​

63

using (var sqlConnection = new SqlConnection(\_connectionString))

64

{

65

sqlConnection.Open();

66

using (var sqlCommand = sqlConnection.CreateCommand())

67

{

68

sqlCommand.CommandText = "SELECT \* FROM [Stocks]";

69

​

70

using (var sqlDataReader = sqlCommand.ExecuteReader())

71

{

72

while (sqlDataReader.Read())

73

{

74

var code = sqlDataReader.GetString(sqlDataReader.GetOrdinal("Code"));

75

var name = sqlDataReader.GetString(sqlDataReader.GetOrdinal("Name"));

76

var price = sqlDataReader.GetDecimal(sqlDataReader.GetOrdinal("Price"));

77

​

78

\_stocks.Add(new Stock { Symbol = code, Name = name, Price = price });

79

}

80

}

81

}

82

}

83

​

84

return \_stocks;

85

}

86

}

As you can see from the code, only an initial select is performed to retrieve and populate the grid. Subsequent updates are done using the SqlTableDependency event handler: its receives a model object containing info about the record changed that we use to update our cache.