.NET Framework SVFs are implemented as methods on a class in a .NET Framework assembly. The input parameters and the type returned from a SVF can be any of the scalar data types supported by SQL Server, except varchar, char, rowversion, text, ntext, image, timestamp, table, or cursor. SVFs must ensure a match between the SQL Server data type and the return data type of the implementation method. For more information about type conversions.

When implementing a .NET Framework SVF in a .NET Framework language, the SqlFunction custom attribute can be specified to include additional information about the function. The SqlFunction attribute indicates whether or not the function accesses or modifies data, if it is deterministic, and if the function involves floating point operations.

Scalar-valued user-defined functions may be deterministic or non-deterministic. A deterministic function always returns the same result when it is called with a specific set of input parameters. A non-deterministic function may return different results when it is called with a specific set of input parameters.

Note
Do not mark a function as deterministic if the function does not always produce the same output values, given the same input values and the same database state. Marking a function as deterministic, when the function isn't truly deterministic can result in corrupted indexed views and computed columns. You mark a function as deterministic by setting the IsDeterministic property to true.

Table-Valued Parameters

Table-valued parameters (TVPs), user-defined table types that are passed into a procedure or function, provide an efficient way to pass multiple rows of data to the server. TVPs provide similar functionality to parameter arrays, but offer greater flexibility and closer integration with Transact-SQL. They also provide the potential for better performance. TVPs also help reduce the number of round trips to the server. Instead of sending multiple requests to the server, such as with a list of scalar parameters, data can be sent to the server as a TVP. A user-defined table type cannot be passed as a table-valued parameter to, or be returned from, a managed stored procedure or function executing in the SQL Server process. For more information about TVPs.

**Example of a CLR Scalar-Valued Function**

Here is a simple SVF that accesses data and returns an integer value:

C#  
```csharp
using Microsoft.SqlServer.Server;
using System.Data.SqlClient;
```
The first line of code references Microsoft.SqlServer.Server to access attributes and System.Data.SqlClient to access the ADO.NET namespace. (This namespace contains SqlClient, the .NET Framework Data Provider for SQL Server.)

Next, the function receives the SqlFunction custom attribute, which is found in the Microsoft.SqlServer.Server namespace. The custom attribute indicates whether or not the user-defined function (UDF) uses the in-process provider to read data in the server. SQL Server does not allow UDFs to update, insert, or delete data. SQL Server can optimize execution of a UDF that does not use the in-process provider. This is indicated by setting DataAccessKind to DataAccessKind.None. On the next line, the target method is a public static (shared in Visual Basic .NET).

The SqlContext class, located in the Microsoft.SqlServer.Server namespace, can then access a SqlCommand object with a connection to the SQL Server instance that is already set up. Although not used here, the current transaction context is also available through the System.Transactions application programming interface (API).
Most of the lines of code in the function body should look familiar to developers who have written client applications that use the types found in the System.Data.SqlClient namespace.

[C#]

```csharp
using (SqlConnection conn = new SqlConnection("context connection=true"))
{
    conn.Open();
    SqlCommand cmd = new SqlCommand(
        "SELECT COUNT(*) AS 'Order Count' FROM SalesOrderHeader",
        conn);
    return (int) cmd.ExecuteScalar();
}
```

[Visual Basic]

```vbnet
Using conn As New SqlConnection("context connection=true")
    conn.Open()
    Dim cmd As New SqlCommand( _
        "SELECT COUNT(*) AS 'Order Count' FROM SalesOrderHeader", conn)
    Return CType(cmd.ExecuteScalar(), Integer)
End Using
```

The appropriate command text is specified by initializing the SqlCommand object. The previous example counts the number of rows in table SalesOrderHeader. Next, the ExecuteScalar method of the cmd object is called. This returns a value of type int based on the query. Finally, the order count is returned to the caller.

If this code is saved in a file called FirstUdf.cs, it could be compiled into as assembly as follows:

[C#]

```csharp
csc.exe /t:library /out:FirstUdf.dll FirstUdf.cs
```

[Visual Basic]

```vbnet
vbc.exe /t:library /out:FirstUdf.dll FirstUdf.vb
```

**Note**

/t:library indicates that a library, rather than an executable, should be produced. Executables cannot be registered in SQL Server.

**Note**

Visual C++ database objects compiled with /clr:pure are not supported for execution on SQL Server. For example, such database objects include scalar-valued functions.

The Transact-SQL query and a sample invocation to register the assembly and UDF are:
CREATE ASSEMBLY FirstUdf FROM 'FirstUdf.dll';
GO

CREATE FUNCTION CountSalesOrderHeader() RETURNS INT
AS EXTERNAL NAME FirstUdf.T.ReturnOrderCount;
GO

SELECT dbo.CountSalesOrderHeader();
GO

Note that the function name as exposed in Transact-SQL does not need to match the name of the target public static method.