

*Copyright © 2004 by The McGraw-Hill Companies, Inc. (Publisher). All rights reserved.*

*www.osborne.com*

*SQL Server 2005 New Features*

*By Michael Otey*

*ISBN 0-007-222776-1*

*400 pages*

*$39.99*

*Coming soon in 2004*

*This is a limited edition preview of content from SQL Server 2005 New Features (ISBN:*

*0072227761). The content is based on the current beta release at the time of writing, and*

*is not a comprehensive representation of the new feature set that will be in the final*

*release of the software and book. We have made every effort to make sure that there are*

*not errors in the text or code as of the writing. However, all of the content is subject to*

*change based on the final release of the software and further editing.*

*Information has been obtained by the Publisher from sources believed to be reliable.*

*However, because of the possibility of human and mechanical error by our sources,*

*Publisher, or others, Publisher does not guarantee to the accuracy, adequacy, or*

*completeness of any information included in this work and is not responsible for any*

*errors or omissions or the results obtained from the use of such information.*

Summary of Contents

Part I: Database Administration Features

Chapter 1: Database Administration and Management Features

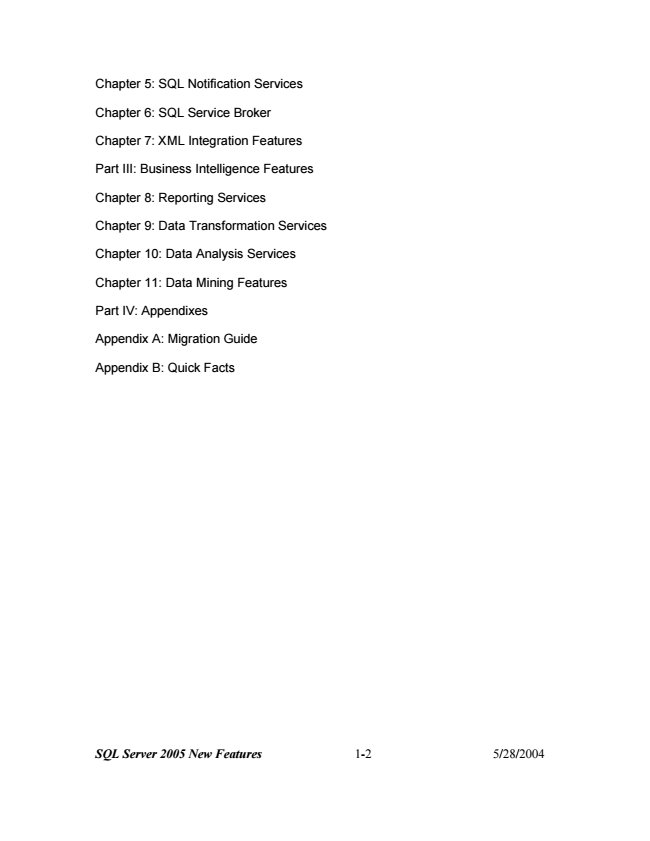
Chapter 2: Architecture and Storage Engine Features

Chapter 3: Availability and Recovery Features

Part II: Database Development Features

Chapter 4: Programmability Features

***SQL Server 2005 New Features 1-1 5/28/2004***

******

Chapter 5: SQL Notification Services

Chapter 6: SQL Service Broker

Chapter 7: XML Integration Features

Part III: Business Intelligence Features

Chapter 8: Reporting Services

Chapter 9: Data Transformation Services

Chapter 10: Data Analysis Services

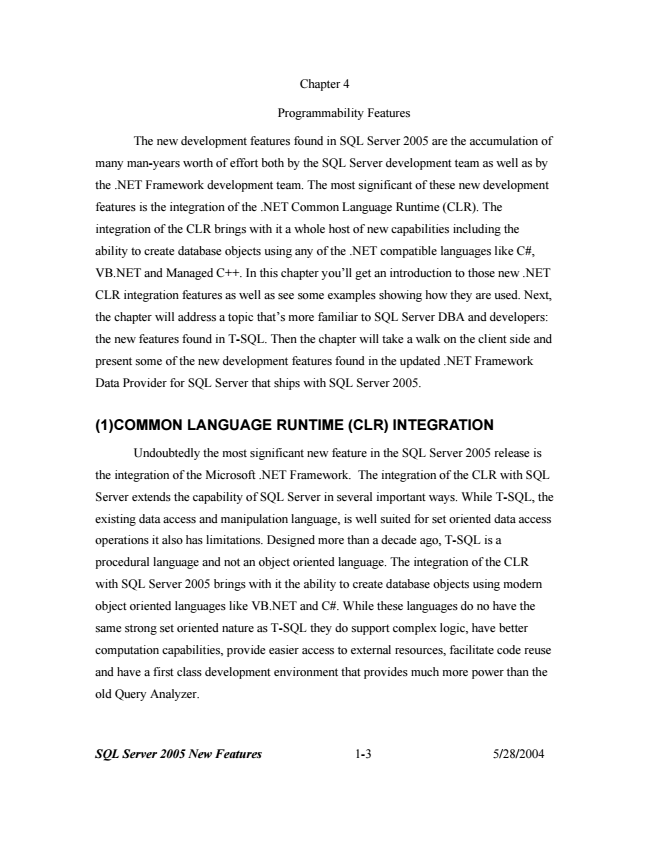
Chapter 11: Data Mining Features

Part IV: Appendixes

Appendix A: Migration Guide

Appendix B: Quick Facts

***SQL Server 2005 New Features 1-2 5/28/2004***

******

Chapter 4

Programmability Features

The new development features found in SQL Server 2005 are the accumulation of

many man-years worth of effort both by the SQL Server development team as well as by

the .NET Framework development team. The most significant of these new development

features is the integration of the .NET Common Language Runtime (CLR). The

integration of the CLR brings with it a whole host of new capabilities including the

ability to create database objects using any of the .NET compatible languages like C#,

VB.NET and Managed C++. In this chapter you’ll get an introduction to those new .NET

CLR integration features as well as see some examples showing how they are used. Next,

the chapter will address a topic that’s more familiar to SQL Server DBA and developers:

the new features found in T-SQL. Then the chapter will take a walk on the client side and

present some of the new development features found in the updated .NET Framework

Data Provider for SQL Server that ships with SQL Server 2005.

**(1)COMMON LANGUAGE RUNTIME (CLR) INTEGRATION**

Undoubtedly the most significant new feature in the SQL Server 2005 release is

the integration of the Microsoft .NET Framework. The integration of the CLR with SQL

Server extends the capability of SQL Server in several important ways. While T-SQL, the

existing data access and manipulation language, is well suited for set oriented data access

operations it also has limitations. Designed more than a decade ago, T-SQL is a

procedural language and not an object oriented language. The integration of the CLR

with SQL Server 2005 brings with it the ability to create database objects using modern

object oriented languages like VB.NET and C#. While these languages do no have the

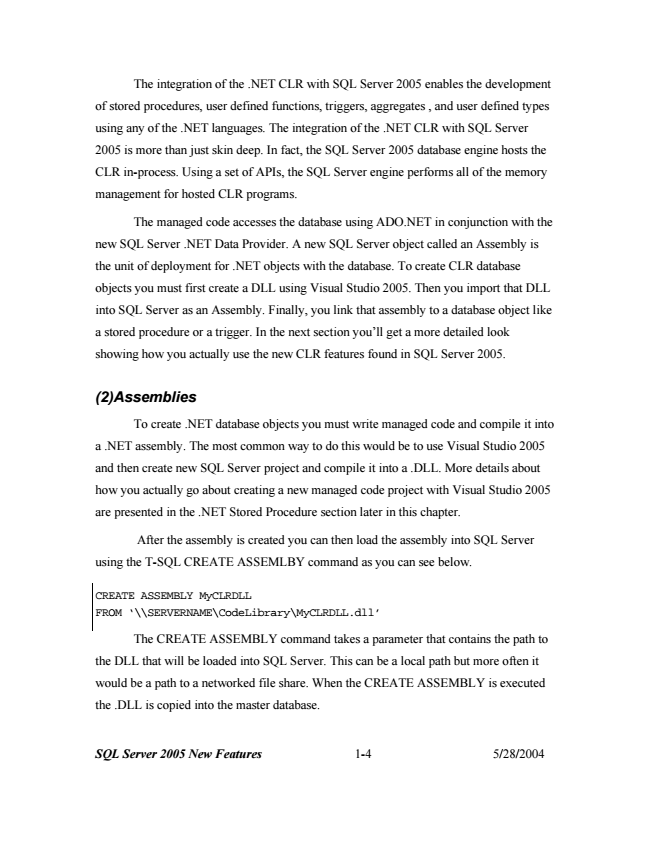
same strong set oriented nature as T-SQL they do support complex logic, have better

computation capabilities, provide easier access to external resources, facilitate code reuse

and have a first class development environment that provides much more power than the

old Query Analyzer.

***SQL Server 2005 New Features 1-3 5/28/2004***

******

The integration of the .NET CLR with SQL Server 2005 enables the development

of stored procedures, user defined functions, triggers, aggregates , and user defined types

using any of the .NET languages. The integration of the .NET CLR with SQL Server

2005 is more than just skin deep. In fact, the SQL Server 2005 database engine hosts the

CLR in-process. Using a set of APIs, the SQL Server engine performs all of the memory

management for hosted CLR programs.

The managed code accesses the database using ADO.NET in conjunction with the

new SQL Server .NET Data Provider. A new SQL Server object called an Assembly is

the unit of deployment for .NET objects with the database. To create CLR database

objects you must first create a DLL using Visual Studio 2005. Then you import that DLL

into SQL Server as an Assembly. Finally, you link that assembly to a database object like

a stored procedure or a trigger. In the next section you’ll get a more detailed look

showing how you actually use the new CLR features found in SQL Server 2005.

***(2)Assemblies***

To create .NET database objects you must write managed code and compile it into

a .NET assembly. The most common way to do this would be to use Visual Studio 2005

and then create new SQL Server project and compile it into a .DLL. More details about

how you actually go about creating a new managed code project with Visual Studio 2005

are presented in the .NET Stored Procedure section later in this chapter.

After the assembly is created you can then load the assembly into SQL Server

using the T-SQL CREATE ASSEMLBY command as you can see below.

CREATE ASSEMBLY MyCLRDLL FROM ‘\\SERVERNAME\CodeLibrary\MyCLRDLL.dll’

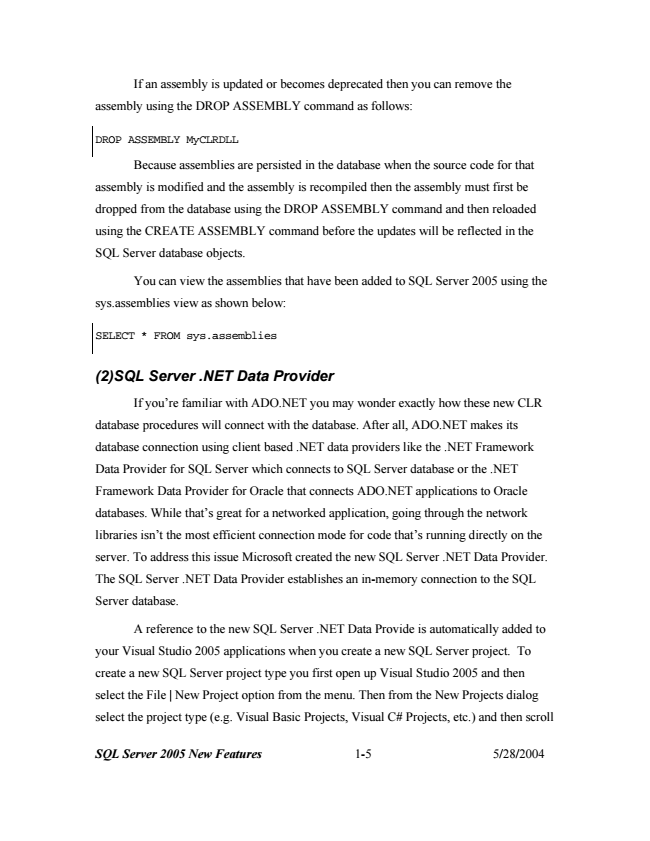
The CREATE ASSEMBLY command takes a parameter that contains the path to

the DLL that will be loaded into SQL Server. This can be a local path but more often it

would be a path to a networked file share. When the CREATE ASSEMBLY is executed

the .DLL is copied into the master database.

***SQL Server 2005 New Features 1-4 5/28/2004***

******

If an assembly is updated or becomes deprecated then you can remove the

assembly using the DROP ASSEMBLY command as follows:

DROP ASSEMBLY MyCLRDLL

Because assemblies are persisted in the database when the source code for that

assembly is modified and the assembly is recompiled then the assembly must first be

dropped from the database using the DROP ASSEMBLY command and then reloaded

using the CREATE ASSEMBLY command before the updates will be reflected in the

SQL Server database objects.

You can view the assemblies that have been added to SQL Server 2005 using the

sys.assemblies view as shown below:

SELECT \* FROM sys.assemblies

***(2)SQL Server .NET Data Provider***

If you’re familiar with ADO.NET you may wonder exactly how these new CLR

database procedures will connect with the database. After all, ADO.NET makes its

database connection using client based .NET data providers like the .NET Framework

Data Provider for SQL Server which connects to SQL Server database or the .NET

Framework Data Provider for Oracle that connects ADO.NET applications to Oracle

databases. While that’s great for a networked application, going through the network

libraries isn’t the most efficient connection mode for code that’s running directly on the

server. To address this issue Microsoft created the new SQL Server .NET Data Provider.

The SQL Server .NET Data Provider establishes an in-memory connection to the SQL

Server database.

A reference to the new SQL Server .NET Data Provide is automatically added to

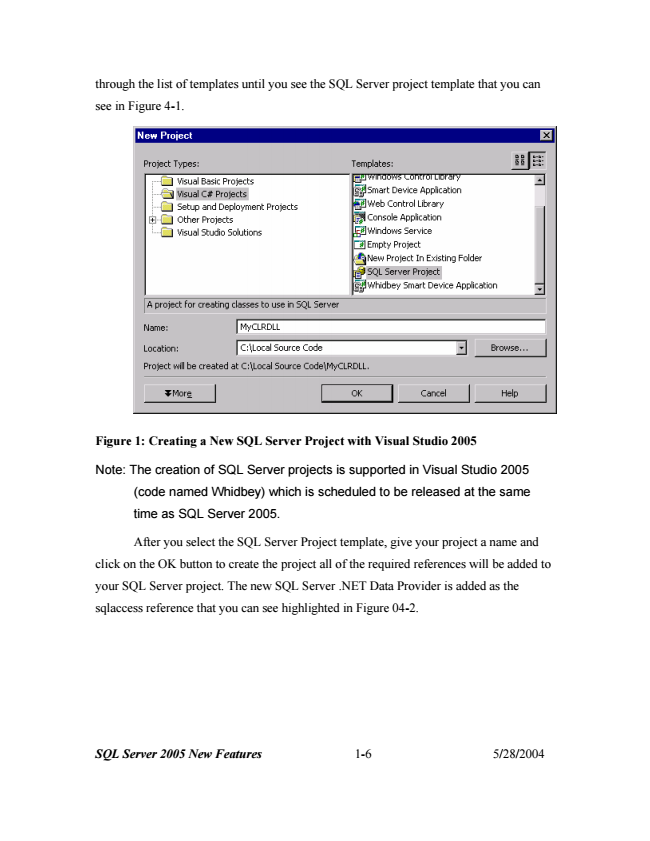
your Visual Studio 2005 applications when you create a new SQL Server project. To

create a new SQL Server project type you first open up Visual Studio 2005 and then

select the File | New Project option from the menu. Then from the New Projects dialog

select the project type (e.g. Visual Basic Projects, Visual C# Projects, etc.) and then scroll

***SQL Server 2005 New Features 1-5 5/28/2004***

******

through the list of templates until you see the SQL Server project template that you can

see in Figure 4-1.

**Figure 1: Creating a New SQL Server Project with Visual Studio 2005**

Note: The creation of SQL Server projects is supported in Visual Studio 2005

(code named Whidbey) which is scheduled to be released at the same

time as SQL Server 2005.

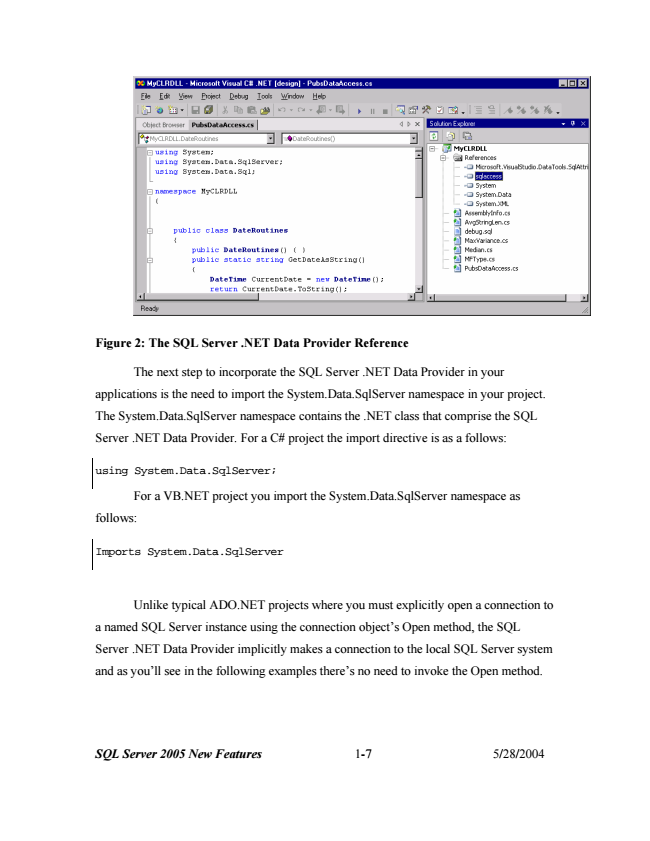
After you select the SQL Server Project template, give your project a name and

click on the OK button to create the project all of the required references will be added to

your SQL Server project. The new SQL Server .NET Data Provider is added as the

sqlaccess reference that you can see highlighted in Figure 04-2.

***SQL Server 2005 New Features 1-6 5/28/2004***

******

***SQL Server 2005 New Features 1-7 5/28/2004***

**Figure 2: The SQL Server .NET Data Provider Reference**

The next step to incorporate the SQL Server .NET Data Provider in your

applications is the need to import the System.Data.SqlServer namespace in your project.

The System.Data.SqlServer namespace contains the .NET class that comprise the SQL

Server .NET Data Provider. For a C# project the import directive is as a follows:

using System.Data.SqlServer;

For a VB.NET project you import the System.Data.SqlServer namespace as

follows:

Imports System.Data.SqlServer

Unlike typical ADO.NET projects where you must explicitly open a connection to

a named SQL Server instance using the connection object’s Open method, the SQL

Server .NET Data Provider implicitly makes a connection to the local SQL Server system

and as you’ll see in the following examples there’s no need to invoke the Open method.



***(2).NET Stored Procedures***

Stored procedures are one of the most likely database objects that you’ll want to

create using one of the managed .NET languages because stored procedures often contain

complex logic and embody business rules that are difficult to express in T-SQL. The

following example illustrates the source code required to create a simple CLR stored

procedure.

using System; using System.Data.SqlServer; namespace MyCLRDLL {

/// <summary> /// SQL Server CLR SP that returns the SalesPerson count. /// </summary> public class AdWDataAccess {

public AdWDataAccess(){} public static int GetSalesPersonCount() {

int iRows; SqlCommand sqlCmd = SqlContext.GetCommand(); sqlCmd.CommandText = "select count(\*) as 'Sales Person Count' "

+ "from Sales.SalesPerson"; iRows = (int)sqlCmd.ExecuteScalar(); return iRows; } } }

The first important point to note in this code is the directive that imports the

System.Data.SqlServer namespace. This enables the MyCLRDLL project to use the SQL

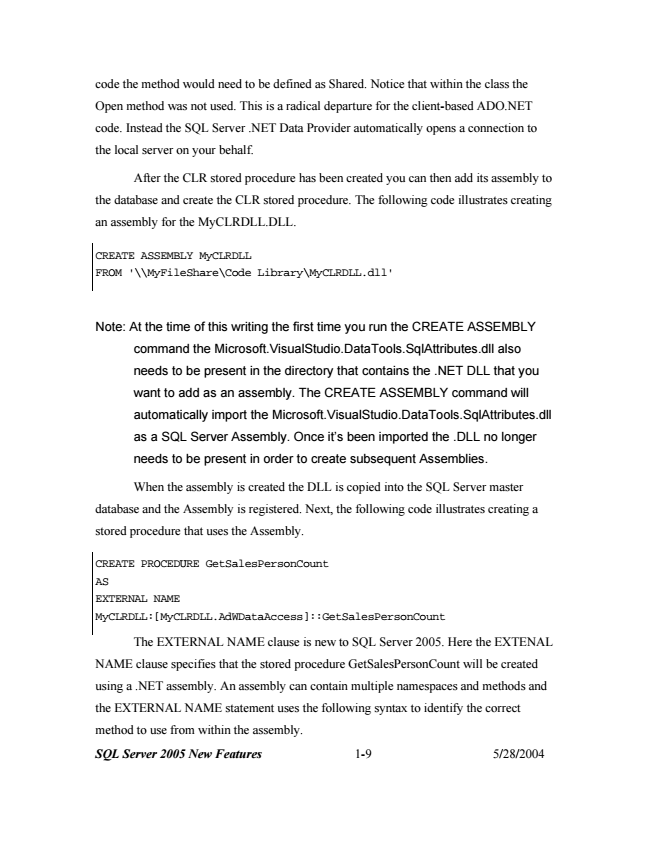
Server .NET Data Provider without always needed to reference the fully qualified name.

Next, you can see that the namespace for this assembly is named MyCLRDLL and that

the assemblies’ class name is AdWDataAccess. This class contains a static method

named GetSalesPersonCount. For C# the method must be defined as static. For VB.NET

***SQL Server 2005 New Features 1-8 5/28/2004***

******

code the method would need to be defined as Shared. Notice that within the class the

Open method was not used. This is a radical departure for the client-based ADO.NET

code. Instead the SQL Server .NET Data Provider automatically opens a connection to

the local server on your behalf.

After the CLR stored procedure has been created you can then add its assembly to

the database and create the CLR stored procedure. The following code illustrates creating

an assembly for the MyCLRDLL.DLL.

CREATE ASSEMBLY MyCLRDLL FROM '\\MyFileShare\Code Library\MyCLRDLL.dll'

Note: At the time of this writing the first time you run the CREATE ASSEMBLY

command the Microsoft.VisualStudio.DataTools.SqlAttributes.dll also

needs to be present in the directory that contains the .NET DLL that you

want to add as an assembly. The CREATE ASSEMBLY command will

automatically import the Microsoft.VisualStudio.DataTools.SqlAttributes.dll

as a SQL Server Assembly. Once it’s been imported the .DLL no longer

needs to be present in order to create subsequent Assemblies.

When the assembly is created the DLL is copied into the SQL Server master

database and the Assembly is registered. Next, the following code illustrates creating a

stored procedure that uses the Assembly.

CREATE PROCEDURE GetSalesPersonCount AS EXTERNAL NAME MyCLRDLL:[MyCLRDLL.AdWDataAccess]::GetSalesPersonCount

The EXTERNAL NAME clause is new to SQL Server 2005. Here the EXTENAL

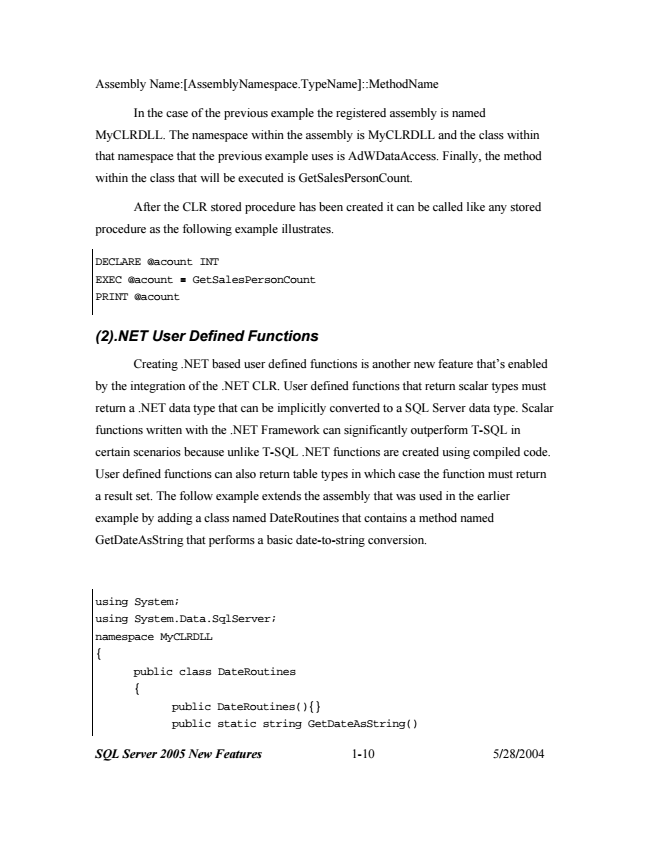
NAME clause specifies that the stored procedure GetSalesPersonCount will be created

using a .NET assembly. An assembly can contain multiple namespaces and methods and

the EXTERNAL NAME statement uses the following syntax to identify the correct

method to use from within the assembly.

***SQL Server 2005 New Features 1-9 5/28/2004***

******

Assembly Name:[AssemblyNamespace.TypeName]::MethodName

In the case of the previous example the registered assembly is named

MyCLRDLL. The namespace within the assembly is MyCLRDLL and the class within

that namespace that the previous example uses is AdWDataAccess. Finally, the method

within the class that will be executed is GetSalesPersonCount.

After the CLR stored procedure has been created it can be called like any stored

procedure as the following example illustrates.

DECLARE @acount INT EXEC @acount = GetSalesPersonCount PRINT @acount

***(2).NET User Defined Functions***

Creating .NET based user defined functions is another new feature that’s enabled

by the integration of the .NET CLR. User defined functions that return scalar types must

return a .NET data type that can be implicitly converted to a SQL Server data type. Scalar

functions written with the .NET Framework can significantly outperform T-SQL in

certain scenarios because unlike T-SQL .NET functions are created using compiled code.

User defined functions can also return table types in which case the function must return

a result set. The follow example extends the assembly that was used in the earlier

example by adding a class named DateRoutines that contains a method named

GetDateAsString that performs a basic date-to-string conversion.

using System; using System.Data.SqlServer; namespace MyCLRDLL {

public class DateRoutines {

public DateRoutines(){} public static string GetDateAsString()

***SQL Server 2005 New Features 1-10 5/28/2004***