

Managing Developers with the Visual Studio .NET tools suite

William Vaughn

Visual Studio .NET incorporates tools for both the developer and other, more important members of the team—namely the database administrator.

This month I mentored a local company that's starting a comprehensive rewrite of their existing PowerBuilder application. They have a lot of work to do, but after two days of consultation I began to realize that they had already solved some of the more fundamental issues facing companies all over the world. I first met with developers and their managers who were anxious to get to the technical details ironed out. Not long after I also met a team of database administrators anxious to make sure that the new Visual Studio .NET tools would not cripple their ability to control the integrity and stability of the company's databases. It was very refreshing. I knew that no matter which type of architecture they choose, they had already made a big forward step toward a stable, successful system.

Visual Studio .NET and Productivity

Easy-to-use tools are a great benefit to productivity and developer efficiency and at the same time a pathway to chaos. In some ways, providing comprehensive tools to developers can be like passing out Swiss Army knives at a Cub Scout meeting. Sure, lots of developers are fully capable of handling the twenty-two blades exposed at the press of a button, but not all of these development teams have the experience or discipline needed to make sure that the data structures, procedures and systems they're creating work well (if at all) with the other pieces of the system architecture.

Since Visual Studio .NET makes it so easy to create databases, tables, views, stored procedures and even SQL Server functions, you would think that developers and their managers would be rejoicing. Well, they are—at least until the database administrators make them aware that these tools can lead to an unstable development environment and a confused, demoralized development team.

Depending on the size and organization of a company, an application development team can be a part-time individual or a full-time highly coordinated and well seasoned army of architects, managers, technical writers, coders, database administrators, UI designers, testers and support professionals. An application developer might be college-trained with years of applicable experience in a variety of skill areas including user interface design, coding, database design, SQL coding and more. On the other hand, I also see too many applications created by well-meaning individuals that have no formal (and apparently little informal) training in computer science. Yes, these folks might have graduated top of their class at a big-name school, but still have no idea how successful, high-performance, and scalable applications are designed, created, tested, deployed, documented and supported. These folks assume that the tools they are using will somehow deal with these issues at the press of a button. But I don't want to focus on these well-meaning "paradevelopers", but on the teams found in many companies large and small that have a firmer grasp on how to create a successful application.

So, for purposes of this article, I'm going to focus on three fairly general divisions of these teams—those that create and code the application, those that manage the database and those that manage, document and support the overall system. There are lots of other ways to split up the work required to build an application and managers usually assign tasks based on the skills of specific individuals and how they work together with others. Sure, as often as not some developers wear many hats—especially in smaller companies. Some design UI, code all or part of the application and write SQL, while others manage the actual DBMS—backing it up, making sure the stored procedures are written correctly, managing indexes and relationships, and preventing anyone (even the developers) from damaging or corrupting the DBMS. In my experience, getting these individuals and teams to work together without killing one another can be quite a challenge; and to this end I'm going to talk about how Visual Studio .NET can help teams of developers not only survive the process of creating applications, but remain friends after the product ships.

Getting started

Virtually all successful applications began with a good design. I've learned that every hour spent in the initial design phase of a project pays off many times over in easier coding and implementation later in the development cycle. When developers work with a clear specification, there are fewer situations where the developer feels the need to "invent" a new component that does not mesh with the larger mechanism. A tangible benefit to a good

design is a better understanding of the application structure (the component parts) and more importantly, how these pieces fit together—the seams where the pieces touch with other pieces. For example, one of the challenges development teams face has to do with COM DLLs. When developing COM-based applications, a number of “contracts” are made between the teams. One of the most important of these agreements is that once a COM component (a DLL) is created, the interfaces are cast in stone—they cannot (or should not) be changed. This means that later, when the component is “improved” or repaired, these interfaces still cannot change. Sure, new interfaces can be added, but in order for the object to be shared and reused, it’s important that the old interfaces are still supported so code that was created with the old version continues to work. The problem is that all through the development cycle as components evolve, their duties change, their functionality changes and their interfaces to the outside world change—constantly. This makes it especially hard on development teams to leverage the work of other teams’ components. One way to address this problem is to design the components and their interfaces first. This way, developers know what to implement and what components and interfaces they can depend on for “outside” functionality.

Visual Studio .NET includes several ways to assist in the design phase of an application. First, the Enterprise Architect version includes Microsoft Visio®-based Unified Modeling Language (UML) modeling tools to specify and communicate application architecture and functionality. Visio can also be used to visually specify and communicate application architecture by generating skeleton code to create starting implementations, and use reverse-engineering capabilities to document existing implementations in all Visual Studio .NET programming languages. In addition, Visio can be used to build a conceptual model of an application. This means Visio also supports full roundtrip engineering with logical and physical data models which enable communication from the business analyst to the database designer. Visio can also be used to help plan the application development project itself—to highlight dependencies and team interrelationships that gate the progress of the tasks. For even larger, more complex applications, architects can draw on Microsoft BizTalk® server to visually orchestrate business processes and graphically link business processes together.

Now-a-days I spend most of my time focused on data—where the languages and tools connect to and query against SQL Server. So I was thrilled when I saw that Visual Studio .NET includes new tools to deal with my development issues. This brings us to the next phase of the discussion—how Visual Studio .NET handles data. Virtually all applications need to access data. Some use databases, some simply access files, or extract data from Web Services or other XML data sources, but more and more we’re required to create sophisticated front-ends to relational databases.

Visual Studio .NET directly supports two of the big three engines Oracle and SQL Server. No, the .NET Framework does not include integrated support for DB2, but there is a new DB2 .NET data provider available. Just how well it integrates with the Visual Studio .NET development tools is not clear, but I expect IBM will make a good effort to make it competitive. Visual Studio .NET also directly supports both ODBC and OLE DB data sources, but I encourage developers to move away from these “One-size-fits-all” OSFA data providers as their performance sucks.

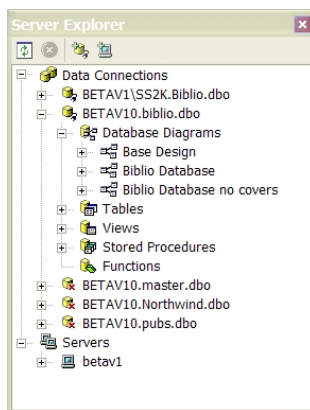


Figure 1: The Server Explorer.

When you first start working with Visual Studio .NET you’ll see a new dockable window—the “Server Explorer”. This window replaces the anemic Data View Window in Visual Studio 6.0. It includes several new features that better support team development of data-centric applications. First, the Server Explorer permits the developer to log on to a target DBMS (hopefully not the production server) and view the Database Diagrams, Tables, Views, Stored Procedures and Functions saved on the system. No, not all providers expose all of these database objects and permit the developer to create new databases, tables, view and stored procedures. Note that

the “Professional” version of Visual Studio .NET does not expose all of the Server Explorer functionality discussed here.

This “Data Connection” is also used to drive the DataAdapter Configuration Wizard (the DACW). The trick here is to coordinate management of all of these DBMS objects with the team that’s responsible for their care. No, I don’t expect database administrators (DBAs) will want just anyone to have rights to add or change these objects, so they will usually turn off permission on these objects to prevent developers from doing anything except viewing the schema and interfaces (the datatypes, stored procedure parameters and return data structures).

Sure, some developers are going to be assigned (or inherit) the responsibility to create and tune the stored procedures and perhaps the entire DBMS structure without the guidance of a competent DBA. However in this case, it’s tougher for others to help as efforts are made to share the workload. All too easily, we could be back to the problems described above when working with constantly changing COM objects. I encourage the use of at least part-time DBAs to make sure that the changes being made to the DBMS make sense and don’t impact other developers’ work.

When developers drill into the DBMS objects exposed by the Server Explorer, they can view the inner structure and attributes of the objects. For example, if you drill into the Tables list, you can see the Authors table returns three columns—Author ID, Author name and the year the author was born. By clicking on any of these data columns the Properties window exposes the individual column attributes. For example, the Au_ID column is defined as an integer. These details are essential when coding applications that need to enforce strong type checking and maintain data integrity.

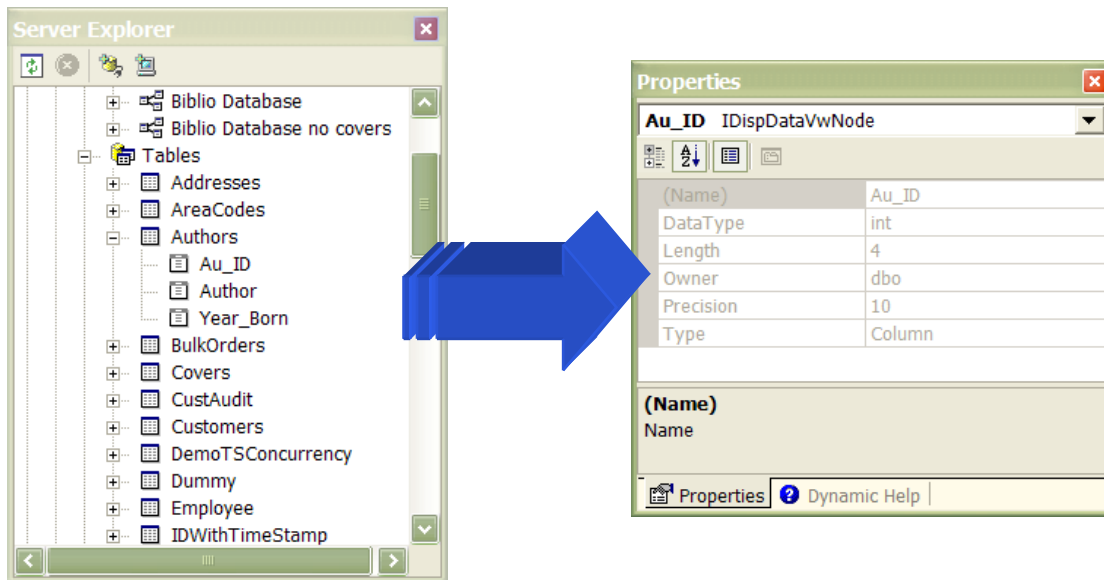


Figure 2: Examining object properties with the Server Explorer

The Server Explorer also supports creation of new DBMS Tables and other objects as well. While the Table creation dialog is a little too much like that used in Microsoft Access, it’s really pretty simple to use. It’s also all too easy to *delete* a database table using this same interface. Although you’re prompted with “Are you sure you want to permanently delete the table from the database?”, at three in the morning, developers can and do things they would rather not talk about.

One of the features carried forward from Visual Studio 6.0 is the ability to view and modify “Database Diagrams”. This gives the developer a change to “see” the database tables and their relationships—as well as manage these relationships, indexes/keys and check constraints. All of these settings, indexes and constraints become a permanent part of the database so, again, this feature can step on the toes of a diligent DBA. For this reason, I think it’s important that the DBA restrict or outright remove rights to make changes to base tables or other objects. Sure, it’s okay to permit developers to “view” these attributes or suggest changes, but not to make wholesale (or worse-yet, subtle) changes.

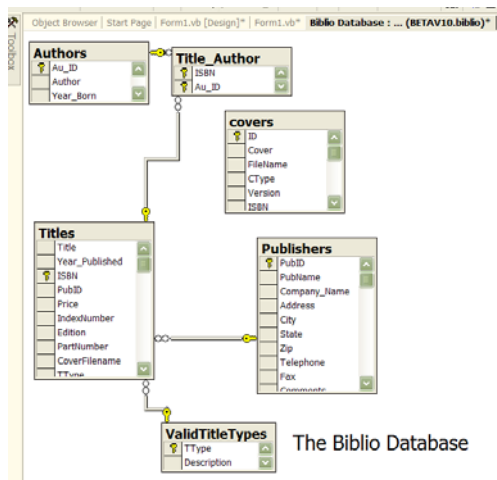


Figure 3: Visual Studio .NET Database Diagram

So what happens when the DBA sets permissions to limit the accessibility of the database objects? Well, as figure 4 shows, once the DBA requires that the developer log on with an account other than SA, the Server Explorer limits visibility to a specific subset of the objects. The DBA chooses those objects to expose based on permissions settings (made in SQL Enterprise Manager). This way DBAs can expose just those stored procedures that the developers should work with. This can be done on an individual basis or by setting up “Roles” which an entire group of developers can join. Again, it’s the DBA that should make this determination. Notice that the Database Diagrams are still exposed. However, when a developer chooses one of these diagrams to view, they are warned that their changes will not be posted to the database. This permits developers to view the database schema but not change it—not unless the DBA has previously granted appropriate permissions to the account.

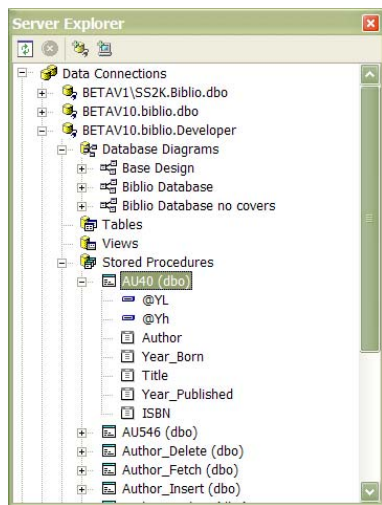


Figure 4: Permission-restricted view of database objects.

Yes, the DBA plays an important role in the development of an application. Without a good DBA, developers will be free to create whatever tables, views, functions and stored procedures they think are necessary, and make changes whenever and however they please. While this might work in a small, intimate development team where the team members actually communicate to each other, in larger development shops, this approach often leads to chaos.

What’s Missing?

So what’s missing? No, as we’ve just seen not all of the tools developers or DBAs need are built into Visual Studio .NET. We saw that the DBA has to use Enterprise Manager to manage permissions or to backup/restore/attach/detach a database or schedule a maintenance job. They will have to start Query Optimizer to examine the query plan or tune the database indexes and Client Configuration Utility to manage netlibs and aliases. However, I think the most important of these missing features is permissions management. That is, if DBAs are going to prevent the mob of developers from ravaging the database, they are going to need the tools

and they simply aren't included in Visual Studio .NET. Perhaps that's for the best. This way the developers can't change the permissions once they are set—at least not without the tools and the SA password.

Building SQL Queries

When developers (or DBAs) need to create SQL queries they have had to depend on their own wits and experience to work out the syntax for the query. Visual Studio 6.0 included a Query Builder tool, but it was tied to the Data Environment Designer. In Visual Studio .NET, this is no longer the case because SQL authors can use either “New Stored Procedure”, “New View” or “New Function” menu options to get access to the Query Builder. Generally, developers must be granted permission to create these objects if they are to be saved. Not only that, but in order to use the Query Builder to generate SQL based on tables or views, they will also need to be granted access to the base tables.

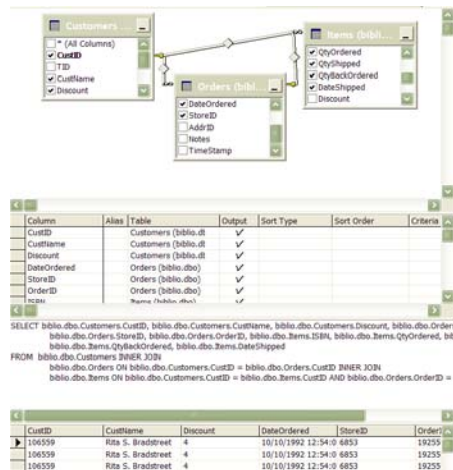


Figure 5: Visual Studio .NET Query Builder

Assuming that the appropriate permissions have been granted, Visual Studio .NET's Query Builder makes it very easy to create SQL queries. These queries can be simple single-table SQL or more complex multiple table JOINS with or without parameters—up to a point. Once the queries get too complex, the Query Builder tends to get confused and the developer must take over coding. The GUI interface is virtually identical to that used by Microsoft Access and Visual Studio 6.0 so developers should have no trouble understanding how it works.

Developers can also use Visual Studio .NET to generate an SQL script for any selected table, view, stored procedure or function. This is especially handy when you want to keep the DBA and the rest of the team informed as to what changes have been made.

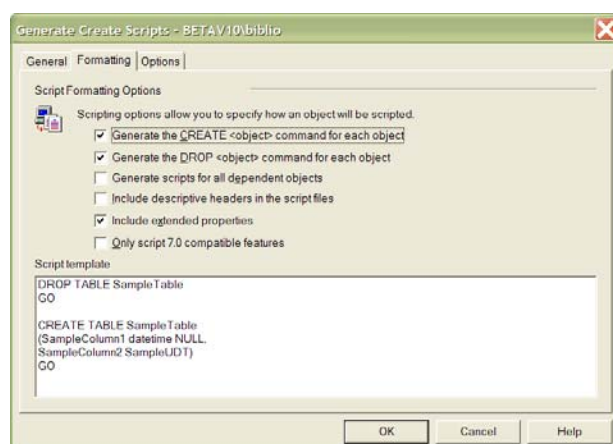
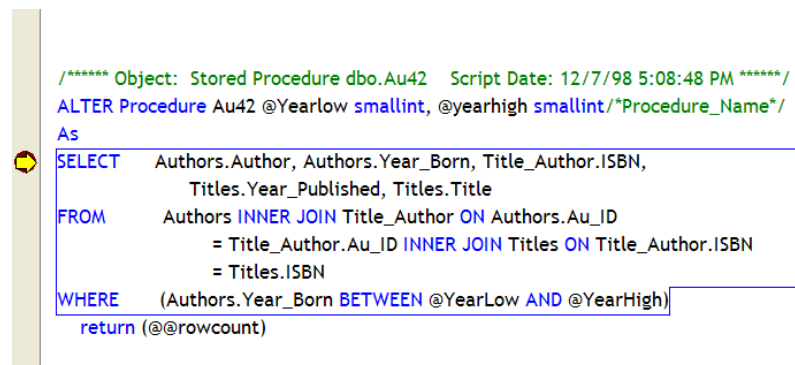


Figure 6: SQL Script Generation

Coding and Debugging Stored Procedures

Up to this point, I expect that I've mentioned stored procedures enough times to make you think that they are an important component to any serious data access application. They are. Developers have found that stored procedures reduce application complexity, increase stability and improve performance. Because of this, for the

last decade I worked with the Microsoft Visual Basic, Visual Studio, MDAC and SQL Server teams to encourage development of better ways to create, manage, execute and debug stored procedures. While Visual Studio 6.0 had a rudimentary form of SQL Server debugging, it was plagued with issues that made it tough to use in a meaningful way—but it was a start. Visual Studio .NET has taken this technology several steps forward to make it easier than ever to incorporate stored procedures into an application and actually step through the stored procedure code while debugging the application. Yes, you have to set several switches and jump through a number of flaming hoops to enable this feature (and we don't have time here to go into the details), but when your code reaches a stored procedure invocation (which might be happening behind the scenes), Visual Studio .NET opens a debug window to let you continue the step-by-step debug process right in the stored procedure code. You can view local and global SQL Server variables and clearly see how the logic of the stored procedure is being executed.



```
/* Object: Stored Procedure dbo.Au42 Script Date: 12/7/98 5:08:48 PM */
ALTER PROCEDURE Au42 @YearLow smallint, @YearHigh smallint /*Procedure_Name*/
As
SELECT Authors.Author, Authors.Year_Born, Title_Author.ISBN,
       Titles.Year_Published, Titles.Title
FROM   Authors INNER JOIN Title_Author ON Authors.Au_ID
       = Title_Author.Au_ID INNER JOIN Titles ON Title_Author.ISBN
       = Titles.ISBN
WHERE  (Authors.Year_Born BETWEEN @YearLow AND @YearHigh)
return (@@rowcount)
```

Figure 7: Stored procedure debugging

No, you don't have to create a routine in code to walk through a stored procedure. Visual Studio .NET permits you to step through any selected stored procedure using the IDE. It prompts you for any needed parameters and permits you to trace through the stored procedure logic a line-at-a-time.

Managing Command Parameters

One of the more time-consuming tasks that data access developers have to face is construction of the Command Parameters collection. This body of code is designed to define each parameter passed in or out of an ad hoc query or (more typically) in or out of a stored procedure. Depending on the data provider, the syntax and rules for each of these Parameters collections varies quite a bit, so developers have to be aware of how parameters are marked in the query, and other not-so-intuitive details. Visual Studio .NET provides several ways to construct these Parameters collections: code them by hand (which is what I do), use the DataAdapter Configuration Wizard (DACW), or write code to invoke the runtime CommandBuilder.

The DACW is executed at design time and extrudes source code that constructs the Parameters according to the rules laid down by the selected .NET Data Provider. Once constructed, developers can use the DACW code directly, or simply extract it, refine it and use it elsewhere in their code. The primary function of the DACW is not to construct a Parameters collection, but to create an updatable DataAdapter that can be used to manage a connection, execute a pre-defined SELECT query (including its parameters) and manage the complex task of updating the data. Since all of the DACW code is visible to the developer, it's easy to modify and refine it to better meet the needs of specific circumstances. The DACW also uses the Query Builder to help generate the initial SelectCommand. It can also automatically generate new stored procedures to perform the "action" queries needed to change the data. Sure, developers can choose existing stored procedures to make the changes to the database.

The CommandBuilder works in a similar way. It constructs simplistic action queries based on a preset DataAdapter SelectCommand, but it does not generate source code that can be tuned and refined. It also executes at runtime where it consumes a round-trip to the server and too many CPU cycles for my taste. If you can't tell, I'm not a fan of the CommandBuilder—which one Microsoft Product Manager called the "Command don't use Builder". I wrote an article published on the MSDN web site that further discusses the failings of the CommandBuilder. (http://www.betav.com/msdn_magazine.htm) if you have any questions as to the wisdom (or lack thereof) of its use.

Visual Source Safe

Visual Studio .NET applications are made up of code and other files of all types that must be managed intelligently. If these files are lost or overlaid by accident, it can cost your development team dearly in lost time, productivity and morale. Microsoft's own development teams use Microsoft Visual Source Safe™ (VSS). This tool helps manage development projects, regardless of the file type. You can store text files, graphics files, binary files, sound files, or video files by saving them to a secure VSS database. Once saved, it's easy (and safe) to share files between developers working on the same or different projects as each file is tagged with a version number and other properties that make it easy to find and manage. Changes made to any file in the VSS database are saved on a version-by-version basis so developers can recover a previous version at any time.

More and more, developers are accessing VSS functions from within their development environment and since VSS is easily integrated with Visual Studio .NET and other development tools, it can ensure that changes each developer makes are properly archived and protected—without interfering with other developer's changes.

The application shown in the slide illustrates how VSS integration exposes icons on each file in the application solution.

Summary

Visual Studio has come a long way over the last few years as Microsoft has adapted its tools to better address the *real* needs of larger development teams. These innovations won't benefit larger development teams much if all of the developers log on as SA and party down on the database schema at will. The best, most productive development teams I've worked with (or lead) all had something in common: discipline. No, I don't mean corporal punishment or a timeout chair facing the corner, but I do mean establishing strict development methodology guidelines and sticking to them. No, these rules were not cast in stone, but we did require a general meeting of all of the developers to change them. That way everyone knew what was expected of them and that they should not have to worry about changes other developers made affecting their own work. Visual Studio .NET goes a long way toward integrating the required tools into the suite. I expect that the next version will go even farther as the Microsoft Visual Studio, SQL Server, Windows and other teams work more and more closely together.

Another final point. The role of the DBA cannot be under-emphasized. It's an important task that requires a person with special skills. Not only does the DBA have to manage an increasingly complex database system, they have to learn to manage unruly mobs of prima donna developers as well. Over the years I've learned to bring flowers at the right time and stay out of the way the rest of the time.