simple-talk

DATA, DEV, ADMIN, CHOP SUEY

LATEST SQL, .NET AND SYS ADMIN ARTICLES, OPINION AND NEWS

Time for a rethink on SQL CLR?

Published Thursday, September 29, 2011 12:10 PM

It is almost seven years since Microsoft announced the sensational news that, with their new SQL Server 2005, you could do data manipulation in Visual Basic or C# as well as SQL. The marketing people got overexcited, stabbing out clichÃ©s from their keyboards such as "new era", and "revolutionary". However, they had run off barking in a different direction to the technologists, missing the whole value of SQLCLR for providing specialized extensions to SQL Server

Unsurprisingly, the technology was misunderstood and many DBAs have never recovered from their initial feelings of dread, still shuddering at the merest mention of the term "CLR function". In the opening passage of his recent review of SQL#, a suite of very useful SQL CLR functions for string manipulation, regular expressions, and more, Grant uses terms such as "throwing your arms up" "screaming", "crying out loud", "nuts" and "stupid", which seems a pretty fair assessment of the attitude of many DBAs toward CLR functions in SQL Server.

Several factors have added to its bad reputation. Some of this is misuse, using SQL CLR for functionality that belongs in the application tier, rather than the database. Security issues have compounded the anathema toward CLRs. Much harm was done by poor implementation, mainly on the part of the coder, but also on the part of Microsoft and SQL Server.

The worst problem is the complexity of writing effective CLR routines for high performance and scalability. The key to success is the constant streaming of data, using **IEnumerable** methods and the **SQLDataReader**, so that SQL Server never has to process too much data at once. The use, instead, of the ADO.NET dataset, and various custom user collections, can make place huge memory demands on SQL Server, involving heavy multi-page memory allocations in the non-buffer pool region. This issue, especially in 32-bit versions of SQL Server, with access to limited amounts of Virtual Address Space, was the cause of paging problems, and specific errors such as "App Domain marked for unload due to memory pressure", or "701" errors, where SQL Server fails to allocate a contiguous region of VAS for an operation.

Unfortunately, SQL Server does not necessarily make it easy for the SQL CLR programmer. SQL CLR functions cannot use a SQLDataReader over the context connection, so the efficient IEnumerable-based streaming simply isn't available and the programmer is forced to use less efficient techniques, or to find ways around the problem, which ultimately involves classifying your assembly as EXTERNAL ACCESS, or possibly even UNSAFE.

If you're someone like Adam Machanic, you persevere. His blog, and various presentations, detail his journey through writing custom IEnumerable classes to enable streaming, creating multiple threads in the SQL CLR to help deal with issues such as "rude aborts" and evolving all this into a powerful "query parallelizer" component.

If all of this is enough to discourage the rest of us from writing our own SQL CLR components, it still doesn't mean we should deny ourselves access to the useful functionality and performance that they can provide. Professionally-written and properly-tested, SQLCLR libraries, such as SQL#, may cause us to refine or qualify the general anti-CLR position. Thorough testing by the likes of the author of SQL#, Solomon Rutzky, has shown that for functions that require heavy computational power, SQL CLR routines have obvious performance advantages over T-SQL. The SQL CLR gives us a "turbo-charged cursor". Any type of calculation that lends itself naturally to cursor-based logic, such as the running total calculation, will run faster as a SQL CLR.

For accessing important CLR libraries such as the Regex functions SQL CLR assemblies are essential. There are a host of specialized requirements where CLR can replace the clunky and antiquated OLE Automation techniques, and there is the added value of allowing new complex data types such as vectors and coordinates. Yes, the technology has its quirks, but CLR libraries are worth having. In fact, unless you have kicked out the Hierarchy functions and geospatial functions, you already probably have CLR libraries installed!

I'd be interested to hear if, and how, you're currently using SQL CLR, or at least if your attitude towards it is softening.

Cheers,

Tony.

by Tony Davis

Mimicking Network Databases in SQL

26 September 2011
by Joe Celko

> Unlike the hierarchical database model, which created a tree structure in which to store data, the network model formed a generalized 'graph' structure that describes the relationships between the nodes. Nowadays, the relational model is used to solve the problems for which the network model was created, but the old 'network' solutions are still being implemented by programmers, even when they are less effective.

"Hey, I can write FORTRAN in any language"
-Anonymous

When I tell people that they are writing sequential file system code or Network Database code in SQL, they insist that they cannot be guilty because they never worked with those kinds of data processing. In a previous article, I showed an example from a forum posting of a tape file implemented in SQL (**Mimicking Magnetic Tape in SQL**). In fact, at the time I am writing this I just saw a forum posting where the poster was having trouble with a procedure:

1. Use an integer (n) as the parameter. It is how many day ago to archive.
2. computed (n) days back from the input date and put it in a local variable.
3. Checked to see if a table for archives existed or not (Hey, people can dismount a tape, so you always checked the tape label before you write to it).
4. Insert rows from the base table into the archive table. Replace "row" with "record" and "table" with "tape" and you have a classic tape file operation. Oh, the old IBM tape labels were "yyddd" from the system clock. There are good Y2K stories about that, but I digress.
5. Delete the inserted rows from the base table (aka Master tape or punch card deck)

See the physical movement of punch cards from deck to another? It never occurred to the poster to simply have a status column in the table, which has 'archive" as a possible value. A simple update everything in the procedure and we do not need a local variable to hold the computation, like we did in COBOL or assembly language.

```
UPDATE Foobars
SET foobar_status
WHERE creation_date <= DATEADD (D, -ABS(@in_day_back), CURRENT_TIMESTAMP);
```

The ABS() is a safety in case we get a negative input.

A similar, but more complicated problem, is seeing a Network Database in disguise. The first problem is that commercial products were all different. And they all used different terms. ANSI X3H2, the Database standards committee, created a CODASYL Standard that became the NDL language. The hope was that NDL would be a reference model to get the product terminology aligned. But nobody used it. In ANSI X3H2, we approved it, forgot it and let it expire while we went to work on SQL. I might have one of the few existing copies of the NDL standard in the world; if anyone has a Gütenberg Bible, I will trade.

Network Database Basics

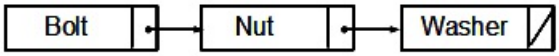
Network Databases use pointers. A lot of current IT people have never worked with pointers. Their languages hide them from the programmer and do all the garbage collection and dirty work for them. And in SQL, the closest thing we have is a reference, which is not a pointer.

In the simplest terms a pointer is a languages construct that resolves to a PHYSICAL address in storage. Notice that I did not say if that was primary or secondary storage. And I did not define what "resolves" means. Yes, it can be primary or secondary storage, with data pages swapping in and out of virtual memory. But resolution is a little trickier. In low level languages, a pointer simply gets you to a position on the data page, but you have no idea *what* it is pointing at. The program decides upon arrival. Higher level languages have pointers that are declared as to their targets, which can be simple data types ("pointer to integer", "pointer to string", etc.) or whole records ("pointer to parts record"). The Network Databases were based on record pointers.

The terminology "link", "parent", "master", "child" and so forth all come from Network Databases. They are not part of RDBMS, but you will find new (or bad) SQL programmers inventing terms like "parent table" or "link table" because they never left a Network Database mindset.

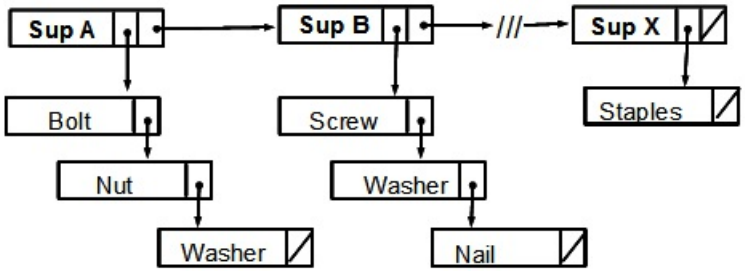
These tools have a NEW() function to create a new, unused pointers as needed and a NIL pointer (not NULL!), which is a pointer to nothing, an end of the chain of links. To navigate the pointer chains, we also have a NEXT function that uses the pointer value to get the next record in the chain.

The target of a pointer can be changed or deleted. Two pointers could have the same target. Basically, anything you can draw with boxes and arrows can be done with pointers. Hey, why don't we draw some boxes and arrows! I will use Chris Date's classic "Suppliers & Parts" Database for these examples.



Here is a simple linked list of parts. You can start at the “Bolt”, invoke the Next() procedure and get the “Nut”; invoke Next() again and get the “Washer”; invoke Next() again and you a message that you hit a NIL, then end of the chain. You are navigating this network, procedurally.

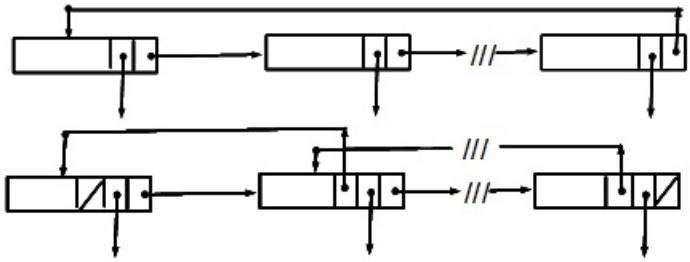
The real power is when you have more pointers in a record. Going back to the suppliers-and-parts. Let's put parts as children to suppliers as parents. That would look like this:



Each node has two pointers. This actually the structure used in the LISP language, with the names CAR (pronounced “car”, short for “contents of A register”) and CDR (pronounced “cudder”, short for “contents of C register”). The weird names come from the original hardware of the first implementation. They are the head and the tail respectively of a list.

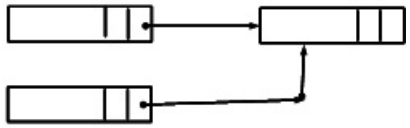
What did you notice about this structure? It is quick and easy find a Supplier by going from tail to tail until you find who you wanted. You then go down the head chain to find the parts from that supplier. Bad news: If I want to find all the suppliers who stock washers, I still have to chase down the parts chain of each supplier. I also cannot go back up the chain.

I can also make a circular list, letting you get back to the start of a list by looping around.



You can get around this problem with a double linked list. It has a extra pointer in each node that goes back to the prior node in the chain.

Different pointers can have the same target, so there is only one occurrence of a node in the system with many references to it.



I can also create nodes with multiple pointers and kinds of things. It can get complicated. And it is fragile. When I delete a node, I have to take pointer and make it the pointer of the prior node the chain and leave the deleted node as an orphan floating in the storage space. If a link is broken, restoring it can be a problem.

Many decades ago I worked for a bank that paid employees by starting a free checking account and making deposits to it. The checking database was a hierarchy which started with wholesale (commercial) and retail (consumer) accounts, then broke retail into internal (employees) and external accounts. The internal accounts were then broken down by department and then you finally got to the actual account. The department for which I worked was sold when banks were no longer allowed to compete with timeshare computing companies.

I went to a branch bank to see if I had gotten my final check (no on-line banking back then). The system crashed. The system crashed every time I asked for the next week. My former department had been deleted and the pointer chain was broken. It never occurred to anyone that a department would disappear while it still had valid accounts in it.

The utility programs for Network Databases had to do garbage collection and restore broken links, or misplaced links (parent Account A gets a bad link and sudden has the children of Account B).

Faking Network Database in SQL

Just as programmers mimicking tape files used IDENTITY for a sequential record number, the “Network Mimic” uses GUIDs, UNIQUEIDENTIFIER and NEWID(). It is no coincidence that NEWID() looks like the NEW() pointer function in a Network model.

Let me look at a forum posting that demonstrates the Network mindset in action. I had to provide the DDL, since the poster did not. In fact, he has a NULL-able column as a Primary Key in his narrative, so it will not compile.

```
CREATE TABLE TBL_Events (
event_id UNIQUEIDENTIFIER NOT NULL PRIMARY KEY,
location_id UNIQUEIDENTIFIER
  REFERENCES TBL_Locations (Location_id),
notes_txt VARCHAR (500));
```

```
CREATE TABLE TBL_Locations (
event_id UNIQUEIDENTIFIER NOT NULL
  REFERENCES TBL_Events (event_id),
location_id UNIQUEIDENTIFIER NULL PRIMARY KEY,
notes_txt VARCHAR (500));
```

Yes, he put the “TBL-” prefix on the table names. Since Network Databases could have many different storage structures, the engine needed to know what was what. In SQL,this is redundant as well as a violation of ISO-1179 rules. We only have tables and they are in limited flavors. Yes, both his tables have the same columns.

To quote the posting: ..

> *>> “Locations is the parent table with a relationship to events through Location ID.” <<*

See the word “parent” instead of the SQL “Referenced” and “Referencing” tables?

To continue: ...

> *>> Location_ID is set to be NewID() on create in Locations, and Event_ID is set to be NewID() on create in events. <<*

This is pointers, not keys. GUIDs are not attributes of either events or locations. They have meaning only after you get to the target of the pointer.

> *>> The table, relationship, and PK format is non-changeable due to organizational policy governing replication. <<*

GUIDs for replication instead of keys is a different issue. It is at a higher level than the schema and treats the whole schema and tables as the unit of work without regard to their content.

> *>> I'm looking for advice on howto define an INSERT TRIGGER that will create a newrowin Events, with the location_id pulled from the parent Locations table, and a newunique event_id. E.g., when (by outside application with no ability to embed SQL) a newLocations record is created, it gets a location_id of 8170daed-92c8-47f1-98ca-147800329686, and the trigger creates a newevent record also with a location_ID of 8170daed-92c8-47f1-98ca-147800329686 and an event_ID of cfed8fe8-b5be-4f78-b366-008672e39637. <<*

Records and not rows. Oh, that's right; Network Databases have records. The mindset always shows up in the terminology you use. What he is trying to do with procedural trigger code is Network navigation. Can you guess where 8170daed-92c8-47f1-98ca-147800329686 is on a map? I use (longitude/ latitude) or HTM (hierarchical triangular mesh) instead; both are industry standard, verifiable with the GPS on my cell phone and reproducible. Likewise, an event should have a name that a human being can understand. And they will both be shorter than GUID.

The idea that an event is BOTH an entity and an attribute of a Locations is absurd. The Events and the Locations have a relationship. What he is trying to build is a double linked list mechanism that would be part of a Network Database engine.

The right way to do this is:

```
-- entity
CREATE TABLE [Events] (
event_id CHAR(15) NOT NULL PRIMARY KEY,
observation_txt VARCHAR (500));
-- entity
CREATE TABLE Locations (
location_htm CHAR(18) NOT NULL PRIMARY KEY,
observation_txt VARCHAR (500));
-- relationship
CREATE TABLE Event_Calendar (
event_id CHAR(15) NOT NULL
  REFERENCES [Events](event_id)
    ON DELETE CASCADE
    ON UPDATE CASCADE,
location_htm CHAR(18) NOT NULL
  REFERENCES Locations(location_htm),
event_start_date DATE NOT NULL,
event_end_date DATE NOT NULL,
  CHECK (event_start_date <= event_end_date),
PRIMARY KEY (event_id, location_htm));
```

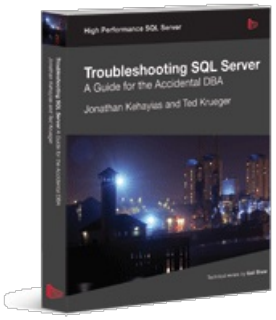
Notice that relationship has its own attributes, such as the time slot in which it occurs. I decided that this is a one-to-one relationship; the poster didn't tell us. I can change this to one-to-many or a many-to-many relationship by changing the PRIMARY KEY. Try doing that in a Network Database! The above schema is easy to search by either location or by event. This is one many reason RDBMS beat out the navigational model of data.

> *"Perfecting oneself is as much unlearning as it is learning."*

-- Edsgar Dijkstra

Troubleshooting SQL Server: A Guide for the Accidental DBA

29 September 2011
by Jonathan Kehayias



Troubleshooting SQL Server: A Guide for the Accidental DBA

Jonathan Kehayias and Ted Krueger

Coming soon in print, eBook and PDF.

We spend much of our working life helping solve SQL Server-related performance problems, hands-on, during consulting, or on online forums. We've seen a few weird-and-wonderful issues but, mainly, it's the same problems and misconceptions time-and-again. This is our attempt to describe, diagnose, and solve the most common problems with SQL Server 2005, 2008, and 2008 R2.

First, we explain a basic approach to troubleshooting, the essential tools, and how rare it is that a problem can be diagnosed by looking at a single data point. The art of troubleshooting SQL Server is the art of first gathering various pieces of information and then assembling the "puzzle" to reveal a complete picture of what is going on inside a server, and so the root of the problem. Next, we explore the areas in which problems arise with alarming regularity:

- High Disk I/O –RAID misconfiguration, inadequate I/O throughput, poor workload distribution, SAN issues, disk partition misalignment and more
- High CPU usage –insufficient memory, poorly written queries, inadequate indexing, inappropriate configuration option settings, and so on
- Memory mismanagement – the advent of 64-bit SQL Server removes the memory allocation "shackles" placed on its 32-bit predecessors, but issues arising from incorrect memory configuration are still common
- Missing indexes – arguably the number one cause of wasteful resource usage in SQL Server
- Blocking – caused mainly by poorly designed databases that lack proper keys and indexing, and applications that apply needlessly restrictive transaction isolation levels
- Deadlocking – covering the Bookmark Lookup deadlock, the Serializable Range Scan deadlock, the Cascading Constraint deadlock and more
- Full transaction logs – lack of log backups, hefty index maintenance operations, long running transaction, problems with replication and mirroring environments, and more.
- Accidentally-lost data – "oops, wrong database!" Let's hope you've got backups%hellip;

In each case, the book describes the most common problems, why they occur, and how they can be diagnosed using tools such as the Performance Monitor, Dynamic Management Views, server-side tracing, and more. Finally, it provides practical solutions for removing root causes, rather than "papering over the cracks".

The steps and techniques described are ones we use day-to-day to troubleshoot real SQL Server performance problems. With them, we hope you can solve performance problems quickly and accurately, and tame your unruly SQL Servers.

The Metro Surfaces

Published Thursday, September 22, 2011 2:34 PM

Our perceptions of Windows 8 are currently based on the live tiles of Windows Phone 7.5 'Mango' and the touch-based 'Metro' User-Interface supported by WinRT. The buzz is of a clean, well-designed, but decidedly retro user-interface based on JavaScript+HTML/CSS, and based in tiles that can be rotated, docked or run full screen but not resized. The focus is on getting the typography right, thereby making the content clear and legible. Amidst all the excitement, .NET, Silverlight and WPF looks a bit passÃ©.

The truth is, of course, different. Windows 8 is still Windows. The 'desktop' hasn't changed, in the sense that nothing has been taken away. Everything will still run in the desktop as it did in Windows 7, but not in 'Metro'.

Metro is for mobile devices. Developers will be able to use the same ALM environment provided by Visual Studio for C# and .NET in order to write mobile apps. The next version of Visual Studio will include project support for Metro apps, and allow Expression to prototype and produce HTML5 Web apps for Windows. The JavaScript editor is now greatly improved, they've added support for HTML5 tags, enhanced the CSS editor and provided better code formatting.

WinRT, (windows Runtime library) is needed for Metro primarily to allow HTML/CSS/Javascript developers simple access to those parts of the operating system that they might require in order to write non-portable Windows applications, and allow these applications to run on ARM processors used in mobile phones as well. However, it is a much more useful part of Windows 8 than just that. It is a managed successor to the Win32 API, written on Object-oriented principles, and able to work with XAML directly. It isn't a replacement for .NET at all. If anything, it replaces COM and Windows Forms. Metro and .NET will both interact with WinRT. For any .NET application, WinRT is just another stack: If it uses WinRT the UI will just run faster, since WinRT uses DirectX to draw the entire user interface.

Having said all that, the presence of WinRT and Metro is uncomfortable for those of us who are committed to developing Silverlight applications, since it reinforces the message that Silverlight isn't for phones or tablets. It looks as if the message from Microsoft is that touch-based Windows mobile applications will be Metro-based, and Microsoft seems very keen on demonstrating how easy it is to convert from Silverlight to Metro. Developers who want to write for Windows mobile devices are going to find it difficult to hedge their bets by adopting a platform that provides portability to iOS or Android. Metro and WinRT isn't going to port easily, even if it adopts open standards. You can't, for example, run either Flash or Silverlight in Metro. It is 'plugin-free'. At the moment, it seems that for a desktop application that has to run on both PC and Mac, Silverlight is still the obvious route. For the Windows Phone and tablet, then it is Metro. For portability? Who knows at this stage?

by Laila

An Introduction to ASP.NET MVC Extensibility

21 September 2011
by Simone Chiaretta

> Because ASP.NET MVC has been designed with extensibility as its design principle; almost every logical step of the processing pipeline can be replaced with your own implementation. In fact, the best way to develop applications with ASP.NET MVC is to extend the system, Simone starts a series that explains how to implement extensions to ASP.NET MVC, starting with the ones at the beginning of the pipeline (routing extensions) and finishing with the view extensions points.

If you are not extending, you are not doing it right: this is the main tenet that you must keep in mind while developing web applications with ASP.NET MVC.

I admit this is a strong statement, but over the next few months I'm going to explain why extending is the right way to develop with ASP.NET MVC, together with explaining when and where you can extend it, and, most importantly, how to do it. But not just yet: to start with, we will look at how a request is processed by the framework, and where during this processing you can replace the default behavior or inject your own logic. This obviously underpins everything we'll be covering in later articles, so let's make sure we understand this clearly.

Prerequisites

Given that we will be discussing several advanced topics in the coming months, it's worth making sure you have a working knowledge of object oriented programming, .NET, ASP.NET, and the basics of ASP.NET MVC. All of the code samples I give you will be written in C#, so you'll need to be comfortable with this language as well. The code samples will also be developed on .NET 4 and Visual Studio 2010.

What is extensibility?

The English dictionary defines extensibility as:

"the capability of being extended"

In computer science terms, a software system is extensible when it can be expanded without the need for changing and recompiling the original source code. This is also referred to as "open/closed principle", taken from Bertrand Meyer's quite well-known writings about object-oriented programming:

"Software entities (classes, modules, functions, etc.) should be open for extension, but closed for modification."

- *"Open for extension"* means that an entity can be made to behave in different ways from the original requirements, and that new requirements can be added.
- *"Closed for modification"* means that the code of such an entity must not be changed once it is declared completed.

If a system does not meet these criteria, it will become very difficult to maintain and evolve, quickly leading to obsolescence.

These criteria are even more important for a commercial software framework that is somehow opinionated and adopts lots of conventions. If it weren't extensible, then it wouldn't be used by developers who didn't like the opinions or conventions adopted.

Because of that, ASP.NET MVC has been designed with extensibility as one of the main objectives; almost every step of the processing pipeline can be replaced with your own implementation. For example, if you don't like the default convention of having actions as public methods of a class, whose name is the name of the controller plus "Controller", you might swap it with your own `IControllerFactory` that implements the convention you think is most suitable for you or your team.

Chances are you've already extended ASP.NET MVC at some point or other. If you've used a view engine other than WebForm View Engine or Razor (such as Spark, for example), or if you've used a IoC container to create your controllers (such as Ninject or Unity), then congratulations! You've already used extensions, albeit other people's.

Where Extensibility Points fit into the Processing Pipeline

At a very high level, the lifecycle of a request in ASP.NET MVC is:

- The browser sends the GET or POST HTTP request;
- The request is inspected and routed to the right controller and the right action;
- The action collects or creates the data that needs to be returned to the browser;
- This data is passed to the view that renders the response, which is sent back to the browser.

Hopefully you already knew this, but to give you a bit more context regarding where we can place all the various extensibility hooks, next we're going to step through the processing pipeline in more detail, and with the help a flow diagram to visually identify it. I know that there's a lot of ground

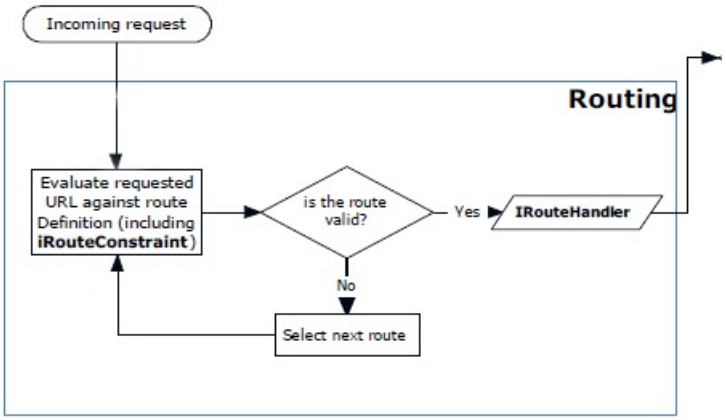
to cover here, and it can appear quite dense if you're not familiar with it, but take your time as you go through this material, and it should all be clear by the end.

The Routing part of the pipeline

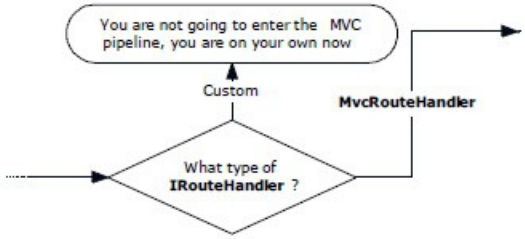
The first macro area of the pipeline is the routing.

> **Note:** The complete flow chart for these steps is available to download as a .PDF from the speech bubble at the top of the article, or from here.

1. Upon receiving the request, the Routing module tries to match the incoming URL with the routing table of the application. The matching is usually done by evaluating the routing URL pattern, or can be made more powerful by specifying Route constraints (IRouteConstraint), allowing you to apply more complex evaluation logic, and even talk to databases or webservices.

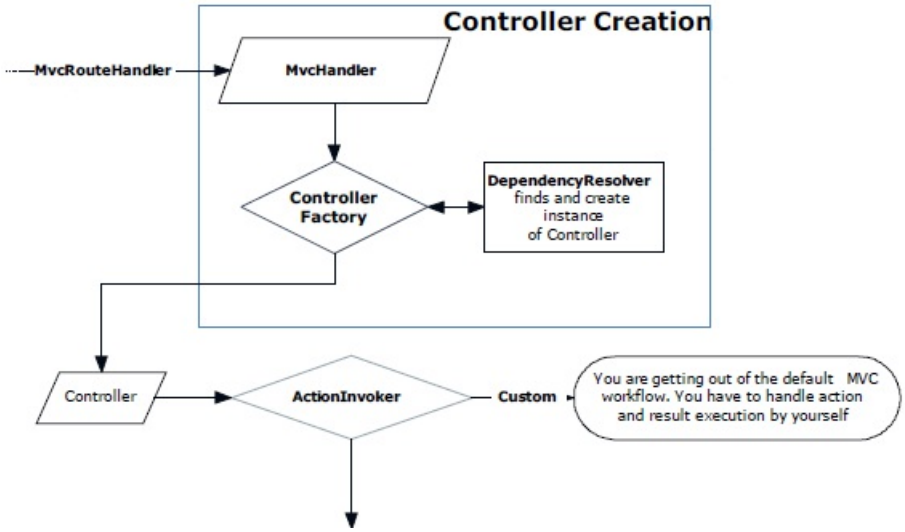


2. Once the Route has been selected, the corresponding `IRouteHandler` is called. The default one calls the `MvcHandler`, which starts the real processing inside ASP.NET MVC. Obviously, if you replace the default handler with something completely different, you also take over the processing of the request, and so all the steps that follow might not apply anymore.



Creation of the Controller

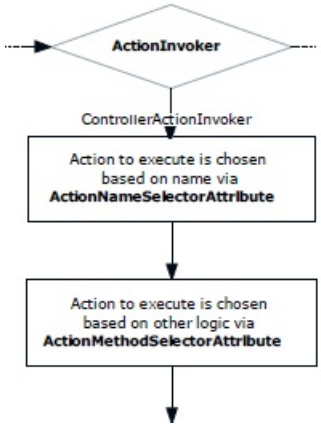
3. Based on the route parameters and the default naming conventions, the IControllerFactory creates an instance of the controller. If an IoC container has been configured, the controller factory retrieves the instance of the controller with the help of the `IDependencyResolver`.



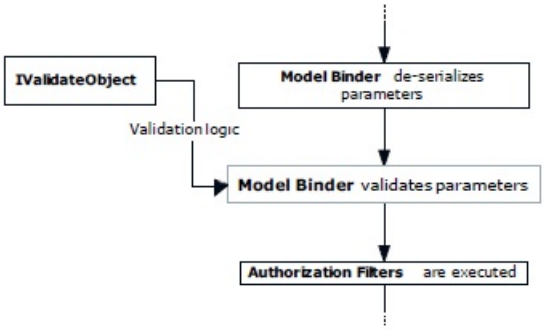
The Action execution

Once the controller has been instantiated, the specified action has to be executed, and this is handled by the `IActionInvoker`. As with the IRouteHandler, in theory you could completely replace this component with your own implementation, but then you will be developing against your own conventions. This is certainly not a problem, but it just means that the next steps will not apply:

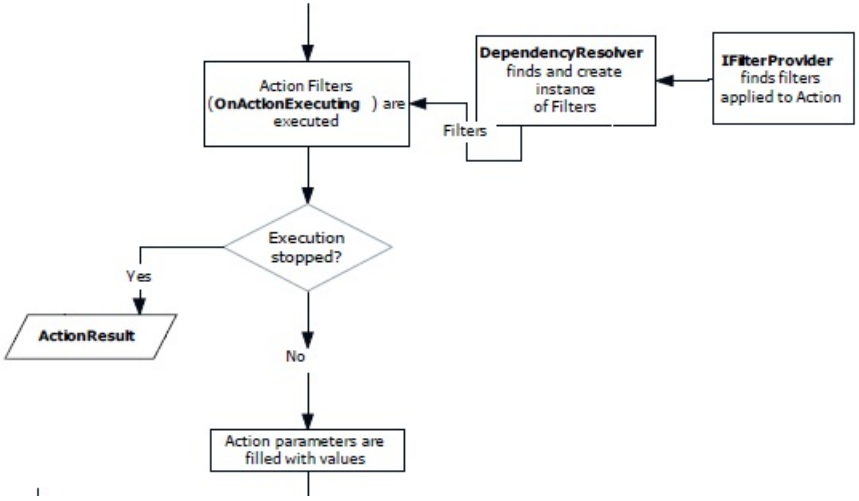
4. The first part of the process involves identifying the method to execute: in the simple scenario, the method which shares the same name as the action is chosen, but this behavior can be modified by applying an attribute of type `ActionNameSelectorAttribute`, which allows you to apply custom logic to this process. If more than one method is found, the correct one is chosen with the help of another kind of attribute applied to the methods: `ActionMethodSelectorAttribute`.



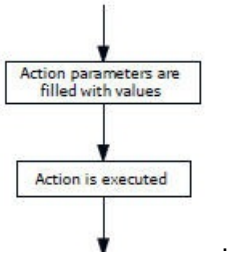
5. If the method selected for execution has one or more parameters, they have to be filled in with the right values. The Model Binder (or rather, the `IModelBinder` interface) is the component responsible for this: it not only deserializes the HTTP request data into CLR objects, but also validates that the values are correct. This is done based on the validation rules (if the default ones are not enough you can write your own rules and validation logic) returned by the Model Validation Provider. Alternatively, another way to validate an object is to implement the `IValidatableObject` for cases where you need a validation rule that acts on the whole object, rather than just individual properties.



6. *to* the action. These filters are discovered via another component that can be customized and replaced: the `IFilterProvider`. In case you are using IoC, the filters are also instantiated using the `IDependencyResolver`. Out of the four types of filters available, two of them are executed before the actual action method: Authorization filters and part of the Action filters (specifically, `OnActionExecuting`). Filters are a very powerful way to implement orthogonal or cross-cutting behaviors that have to be shared among many actions or controllers. Filters can be applied per action, per controller, or even per application.



7. If the filters don't stop the execution the action method is executed and the parameters are filled in with the values that were previously discovered by the Model Binder

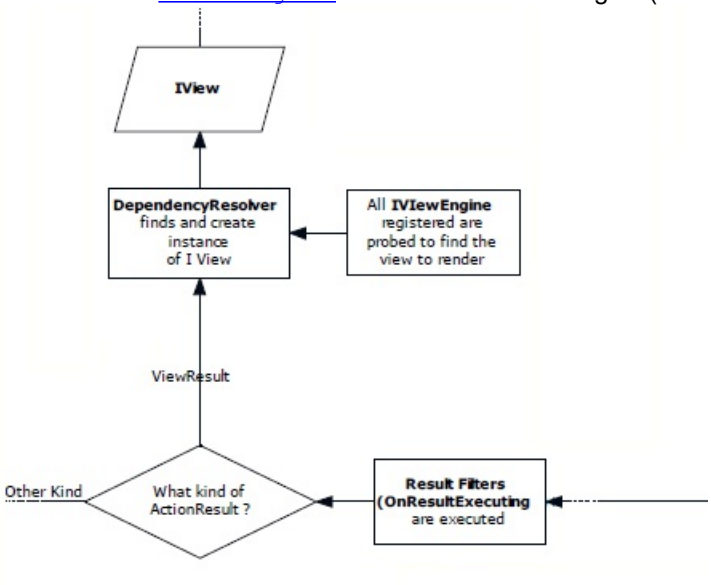


8. When the action method has completed its tasks, it returns a result of type `ActionResult`. This is executed to return the response back to the client that sent the original request. However, there might be other filters that can be applied before the response is returned: the `OnActionExecuted` and the `OnResultExecuting`.

Finally, the View

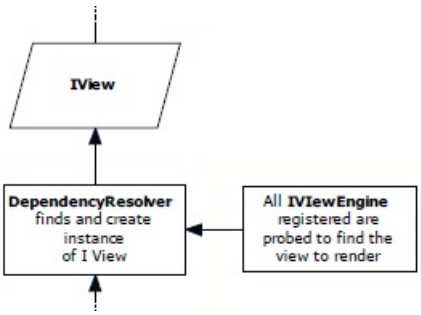
There are many action results: one simply returns a text string, another returns a binary file or a Json formatted result. Yet the most important Action Result is the `ViewResult`, which renders and returns an HTML page to the browser.

9. In ASP.NET MVC 3 there are two standard view engines: the WebForm view engine and the new Razor view engine, and their views are just normal .aspx pages (typically without a code-behind file) or ASP.NET Web Pages. However, more than one view engine can be registered, so the first step in the execution of the View Result involves selecting the appropriate engine to use and the view to render; which is all done via the `IViewEngine` interface of the view engine (which, despite name, takes care of finding the correct view).

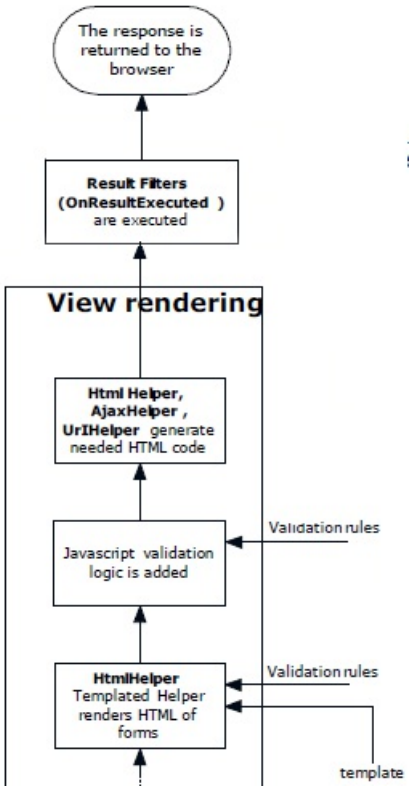


10. Once the correct view engine and the view is chosen, it is instantiated, again with the help of the IDependencyResolver, and the model is then passed to the view.

    The lifetime of a view then depends a lot on what it does; in the simplest case it could just render a bit of text, even without the need of HTML helpers. Alternatively, in a more complex scenario, it might render a form with JavaScript validation. In the latter, more complex case, we actually encounter two new extensibility points: the templated helper… er… templates, and the Model Metadata Provider. These two components work together to help you write less code, automatically creating the form based on the properties of the model object that you have to edit, as well as the metadata attached to it. While the default provider looks at attributes on the model class, if you want to store the metadata somewhere else you can write your own provider that provides metadata based on your own conventions.



11. At this stage, the Model Validation Provider (which we encountered during the action execution) comes onto the stage again. Once the information from the model and the validation rules has been retrieved, client-side validation scripts are automatically added to the page. This means that verifying the length of a string, the value of a number and many other simple requirements can happen without you having to writing a single line of code. Just like the server-side validation that happens during the model binding stage, you can also add your own validation rules on the client side within their JavaScript code.



Another type of validation that happens on the client is **remote validation**; this is needed when the validation rules require access to resources on the server. For example, if you have a form in which you need to check whether a username has already been taken, the validator needs to call back to the server to determine whether the field is valid.

Now, when writing the code for the view, you will use three libraries:

- HtmlHelper
- AjaxHelper
- UrlHelper

They contain helper methods to write input fields, create links based on the routes, AJAX-enabled forms, links and more. Moreover, extending them is very easy, and writing new methods for these libraries, as opposed to writing the code directly inline in the view, is also considered an ASP.NET MVC programming best practice.

Types of extensibility points

Now that we've gone over the process pipeline in more detail, you hopefully have a better idea of what's happening under the hood, and you should also be starting to get a feel for the number of opportunities you have to extend your ASP.NET MVC application. However, as you might have already noticed, not all extensibility points need to be used with the same frequency:

Essentials

Some customizations are integral parts of the development of a web application with ASP.NET MVC, and thus are used in every project. The most common extensibility points in this category are:

- Custom templates for the templated helpers
- Server-side validation rules
- Client-side validation rules
- HTML Helpers

Productivity enhancers

Other extensibility points are not always needed, but if used can help make development more productive and less repetitive:

- A base controller to enforce your own conventions
- AJAX/URL Helpers
- Action/Result Filters

Case-Specific

Next up, there are other extensibility points that you only need to use if you need to solve a specific problem:

- Route Constraint
- Route Handler
- Action Result
- Model Binders
- ActionMethodSelector

Core Extensions

Finally, there are the extensibility points that change a big part of ASP.NET MVC, and which you are not very likely to implement. These might even remove some of the other extensibility points and add their own. Nevertheless, there is a very high chance that you used a custom component for one of these points that was developed by someone else:

- Controller Factory
- Action Invoker
- View Engine
- Model Validation Provider
- Model Metadata Provider

How these Articles will be structured

During the course of these articles, I'm going to explain how to implement your extensions to ASP.NET MVC, starting with the ones at the beginning of the pipeline (routing extensions) and finishing with the view extensions points. I'll focus mainly on the first two categories (“Must Use” and “Productivity Enhancers”), but I'll also briefly touch upon the more rarely implemented ones, where there are useful modifications to be made.

For each point explained, a common structure will be followed:

- why your own custom implementation might be needed
- area of extensibility (routing, controller, views, etc)
- type of extensibility (must use, only when necessary, etc)
- the interfaces that have to be implemented
- what the default implementation (if it exists) does
- sample of already available implementations in the community (if any)
- sample implementation

Thus far we’ve seen an overview of the entire ASP.NET MVC processing pipeline, and should now have a feel for the huge number of opportunities available to extend and customize how the framework behaves. We’ve also seen that not all extensibility hooks are created equal, and there are some which are more useful than others. Now that we’re all on the same page, the next article will cover how the routing process can be extended.

> The free wallchart, 'ASP.NET MVC Pipeline', that goes with this article is available as a PDF file from the speechbubble at the head of the article or from here. It is best printed on an A3 printer.

It's just HTML

Published Thursday, September 22, 2011 10:00 AM

This blog post is based on a lightning talk I gave at the Technical Communication UK conference in Oxford on 22nd September 2011. The theme was 'If I ruled the world', and the topic was 'authoring tools that limit the way we work'.

Before I start, I should note that I'm going to be talking about Author-It here, but this isn't a rant about Author-It. Instead it's an argument that we don't actually need anyone's help authoring tool at all. Also, I'm assuming that you're outputting your content primarily to the web; if you're not, you should already be asking why not, but that's a whole different discussion.

Recent development work on the ANTS Memory and Performance Profiler has required me to do a few things that push the boundaries of the tools and infrastructure that we use at Red Gate. For example:

- The Memory Profiler has the new Instance Categorizer graph, which cannot be described without a screenshot wider than the 630 pixel width currently allowed by our website.
  The solution I adopted was to use jQuery to zoom into the screenshot when the mouse is hovered over it.
- The Performance Profiler has a new demonstration application, which (although very straightforward) takes 41 steps to describe.
  If the description were in simple text form, it would doubtless be off-putting to evaluators. By splitting it into a Flash presentation, with lots of screenshots, the demonstration hopefully doesn't seem to be so much of a chore.

Both of these should be possible in Author-It, our current Help Authoring Tool (HAT), using HTML Snippets. Snippets aren't a great solution though because they require working with raw HTML which doesn't always display in the editor. It's even more of a problem for us, because the XSL transforms applied during our publishing process tend to break unusual things in the Author-It output anyway.

As a result, I started intervening in our publishing system, by manually editing the HTML output between the output of the XSL and putting the page on the website.

Given the numerous problems we have encountered using Author-It (those of you who have followed my colleague Roger Hart @RMH40 on Twitter for a while will already know about these difficulties, so I won't repeat them here), I started to wonder what benefits we gain from it.

Let's take a look at the main features Author-It provides according to its own website and consider how you could replicate these features on the web without the tool:

- *Reuse existing content as you type, prompting you with similar or identical phrases used elsewhere in the library.*
  Server-side includes are easy in most publishing systems on the web.
- *Work in an optional web-based authoring interface, allowing staff anywhere with an internet connection to log in and work.*
  Wikis (for example, MediaWiki, Atlassian Confluence and even SharePoint) do that... though I'm definitely not suggesting using SharePoint for documentation.
- *Manage workflowto assign content tasks, track the results, and viewoverdue tasks*
  If documentation tasks are in the company's existing bug tracking system (at Red Gate we use Atlassian's JIRA), you don't need another tool to manage workflow.
- *Reviewand edit content in real time*
  What does this even mean? That you type a letter on the keyboard and it appears on the screen immediately? Well yes, even Notepad does that.
- *Localize content as it's ready, reducing localization costs*
  We don't localize our products, so this is perhaps a more major consideration for other organizations, but surely web pages could be localized when they are ready too?

Given the lack of a compelling argument in favor of Author-It here, I asked on Twitter what features users most value in a HAT. The answers were:

- *User-defined variables*
  Again, this is trivial in most scripting languages for the web.
- *Style control for online and print*
  The @media type in CSS2 allows you to specify different online and print formats.
- *Code editing*
  Most HTML editors allow you to edit the HTML code manually.
- *Single-sourcing*
  I've always been fascinated by the industry's fascination for single-sourcing. I've never found it useful personally. But anyway, as noted above, single-sourcing is basically a server-side include as far as any web developer is concerned.
- *Link viewer*
  This is the ability to see which pages link to which other ones in your content and, OK, I'll admit that I'm a bit stumped on this one. After publication, Google Advanced Search can do this but to do it before publication you'd need to write a short script to crawl your content on

a test server. It shouldn't be too hard, however.

So, there's possibly one feature for which a HAT is actually useful.

Actually, from my own experience, I can add another two: creating Tables of Content, and the ability to quickly compile the HTML output to CHM or PDF. Neither of these is insurmountable, though.

In conclusion, my thesis is that, if you're outputting primarily to the web, a HAT is basically an impediment to your ability to innovate. Let's just not use them, and replace them with a sensible HTML editor, such as Adobe Dreamweaver. That will allow us to do anything that the web has to offer.

It also opens to door to using proper content management systems, improved meta-data and, for us Author-It users who are reliant on that product's limited implementation of source control, access to proper version management.

Ultimately, all we're doing is putting HTML on the web; and editing and publishing web pages has been a solved problem for well over a decade. This approach allows us to concentrate on the writing (which is our specialism), and not to worry about the tools we use.

> *Although this lightning talk was well received at the conference, I suspect it might get a lot more comment published on the web. Please do give me your feedback. Is there something I've completely missed?*

by Dom Smith
Filed Under: technical communications

Going Beyond the Relational Model with Data

27 September 2011
by Robert Young

SQL is a powerful tool for querying data, and for aggregating it. However, you can't easily use it to draw inferences, to make predictions, or to tease out subtle correlations. To provide ever more sophisticated inferences to businesses, the race is on to combine the power of the relational model with advanced statistical packages. Both IBM and PostGres are ready with solutions. And SQL Server? Hmm...

If you've accepted Codd's rules and you lead a purposeful life, then your concept of data is relatively simple compared to the heathens who still worship at the altar of the flat file. Your schemas are BCNF or better and your SQL isn't polluted with RBAR; just JOINs, DRI, and a view or two.

But, there's more to data than the RM (Relational Model). If data is your addiction of choice, I've got something to tempt you... statistics, embedded with relational databases. A currently prominent meme is "data science", and the publishers are printing to a fare-thee-well. It happens that there can be an integration of the base science of mathematical statistics and relational databases to make a more integrated manifestation of "data science". It was recently reported that R and SQL are the most-used languages of those practicing data science. In practice, that means the analyst dumps some rows from a database, imports them into a session with the statistical package, generates some output, massages this for use by a client program, puts the output in some place, rinse, repeat. "There are more things in heaven and earth, Horatio..."

My goal is to impel you to consider data science as an extension of the relational database, and even to explore the world of inferential statistics. By 'inferential', I mean the kind of statistics that support predictions, as opposed to descriptive statistics, which merely tell you the state of the world you're looking at. I offer not a "how to", but a "can do... and you can do even more".

The way to achieve this is by using statistical packages, or 'stat packs'. Stat packs are to the world of the mathematical statistician what DDL generators are to the professional database developer—a means for the ignorant to wreak great havoc. Having been party to ISO-9000 and Six Sigma implementations, I've seen the results first hand. I don't follow their lead, anointing acolytes as *statistician-lites*; Instead, I encourage readers to invest some time to really understand what goes on in the field of predicting from data.

And if you're going to do predictions from data, do it from the database... I'm not talking about vanilla stat packs talking *to* some database, for they all can do that: Rather, I'm talking about the database calling stat pack routines as it does any other engine function; different, and very intriguing .

Later in this article, I'll be providing an example scenario using PostgreSQL (more commonly known as Postgres), since it has integration support with a stat pack. The stat pack, which has been a decade-long overnight success, is R. The name is just R. It's an open source alternative to S, from the 1970s. Kudos to Joe Conway for writing PL/R, the R procedural language for Postgres. Note that I'm not going to advance tutorials on the constituent parts of the example (mathematical statistics, PostgreSQL, PL/R and R); there are many and the bibliography at the end of this article contains links to resources.

But before that, I'll begin with some background information on the tools we'll use.

Stat packs

All of the four most-known stat packs today have roots at least back to the 70s:

SAS (originally, 'Statistical Analysis System') is the big "kahuna" of the field. SAS is expensive, and by design, intended to be a total data management experience. Because of a decision early on to make the product a one-stop data system, it is not likely to be integrated into any particular RDBMS. SAS Institute Inc is a private company that is highly unlikely to be acquired.

SPSS (originally, 'Statistical Package for the Social Sciences') was bought by IBM, and is being integrated. I expect to see it being promoted for use directly in DB2 at some point. SPSS is known to be more SQL friendly than SAS.

Both SAS and SPSS emerged in the late 1960's on mainframes (not just IBM; there really were others back then), and (along with BMDP Statistical Software), were used primarily in academic research. As an aspiring econometrician, I used them all.

Minitab is the stat pack of choice for ISO-9000 and Six Sigma contract instructors. It provides Shewhart control charts and the like. It is not what we're looking for in our example.

There isn't a SQL Server-centric stat pack that I'm aware of, although XLeratorDB from WestClinTech does provide some support (I've not used it).

So, off we go to R. R is an open source implementation of S (which is commercially offered as S-Plus), which was first developed at Bell Labs in the mid-1970s. What makes R so popular with the data analysis crowd is a very large, and growing, set of functions (available as entities called packages) written mostly in R and easily available. The main repository is CRAN. This is one case where "free" hasn't been the impetus for adoption; it's the large community of users/developers.

The principle difference between R and the others can be summed up, thus: SAS/SPSS/etc. are statistical command languages, while R is a statistical programming language. This difference gives R an advantage when integrating to a RDBMS.

Historically used by engineers and scientists, its host language tends strongly to Python (sorry, C#-users). R can read SQL databases with RODBC and RJDBC R-side drivers. In addition, there are R-specific database drivers, but for (mostly) open source databases: RMySql, RPostgreSQL, RSQLite, and ROracle; no RSQLServer as yet (sorry, SQL Server users). The database drivers allow more or less direct read/write of the database from R under the R-specific protocol DBI. Connecting to the database from an R client session is the usual method of integrating. In our example, we will want to run R functions against the data from the database.

There are a host of online tutorials and suchlike; some are listed in the bibliography, along with the texts I've used. There are a couple of other R books out, but not all that many, curiously.

Here is a review of the code base of R and the contributions: How much of R is written in R: Part 2 Contributed Packages.

Postgres and R

Postgres is, for a reason probably related to Stonebraker's background as a coder, regarded as a programmer's database. The Rules approach is very procedural, and differs markedly from Codd's declarative way; not surprising in that Codd viewed the database from a mathematical and logical stance, and sought to reduce errors with a declarative structure. Folks have been arguing for decades whether declarative logic is better than procedural. Yet the coder folks are forever making frameworks and suchlike that insinuate more of the logic in declarative fashion. However, it seems they're not yet willing to fold their tents and let the relational model win.

Postgres supports external function languages for which wrappers exist, in addition to the languages the base engine supports. This is how R can run in PostgreSQL. Since R is largely implemented in C, the wrapper runs through it. This also implies that any engine that supports C as an external language could also run R, in much the same way PostgreSQL does (hmmm...). My goal isn't to write stat functions in R, but to call the built-ins and graph plotters.

The process of getting R and PostgreSQL up and running is a bit convoluted. The bibliography contains links to the necessary background. In one form or another R has been "embeddable" in PostgreSQL for about a decade.

For completeness: I used Ubuntu 10.04, PostgreSQL 8.4.4, R 2.13, and PL/R 8.3.0.13. The three bits of software are all available for Win and OS/X.

Stat algorithms

One last bit of background... A huge difference between the relational database world and the math stat world is that research in the latter continues to find new algorithms and implementations.

Stat algorithms tend strongly to find their expression in linear algebra, which is to say matrices, so that the data stores have historically been what relational people would possibly call first normal form tables. R has support for syntax which just about does what a relational join does; I expect that the R folks will soon enough complain about "impedance mismatch" and try to join (the verb is merge) in R. Public data, in the US anyway, is nearly always some form of flat file; some R authors even assert that such is the preferred structure. We can produce the join in PostgreSQL and pass it to the R function.

The scenario

To illustrate the possibilities when a stat/graphics pack is embedded in the RDBMS, I want a scenario where one needs to predict some future outcome based on the data. Predicting sales of nappies is boring; albeit lucrative. Let's not do boring.

So, let's say you're a downtrodden political party, just having gotten smacked upside the head with a cricket bat in the last by-elections. And further, let's say that the apparatchiks decide that some sort of data-driven application should be built to avoid the same result in the coming elections. What would such an application look like? And how would one slog through the data to find the predictive equation?

What follows is fabricated for this essay. (No politicians or political operatives were harmed in its production.) What is required here is something in between disinterested prediction (Gallop and FiveThirtyEight and so forth) and campaign strategy (“let's go negative on my opponent and kiss supporters' babies”): we're not interested in simply predicting the outcome, nor will we have the authority to dictate strategy depending on what our application tells us. We want some form of feedback that can be used by the campaign decision makers, and which measures effectiveness of campaign spending, in a “Biggest Bang for The Buck” sort of way. This is the classic optimization problem from first semester calculus, but I won't be writing derivatives.

And I don't get to do the really cool part either: data exploration, which is the general term for spending time with a lump of data to find which general type of analysis to conduct, and having arrived at the analysis to do, working the model through the data. No, instead I'm forced to concoct a set of data which could reasonably be expected in the wild.

All logic begins with assumptions, and I'll make some here. There is, available to the apparatchiks, both public data (the FEC here in the States) and data developed by their own organization. This latter data is, amorphously, expenditures (the source data that ends up at the FEC; their own they have, but opposition data must wait for FEC and may well not be sufficiently timely) and outcomes; perhaps simple polling results; perhaps some focus group results; perhaps some name recognition surveys. Social network data mining is also big these days (although I've not done enough research to know for sure that this could be a data source for outcomes). And so on. In any case, there exists reasonably objective data measuring the electability status of Our Candidates.

I'll make the grand assumption that the organization has other data (again, some mix of public and proprietary) which can be predictive, rather than just retrospective. Such data might be: issue vulnerabilities of candidates (theirs and opposition), cultural biases among the voters, issue recognition level among voters, Q Scores.

The point here is that such data exists, but the predictive ability of any one part of the data is not known a priori. The apparatchiks have a gold mine, but they don't know *exactly* where the gold is, and they have a relatively short time to find it. How to find the gold is the prime question. The gold may, or may not, be the simple act of spending more money on some number of candidates. This is the place to mention that studies have found that winning elections is not highly correlated with expenditures. Spending money smartly, in the Darwinist political sense, is at least as important as spending money.

Defining the objective

First, the apparatchiks must define the objective. The obvious, and naive approach is to measure $$$ spent on candidates, measure outcome (poll numbers, most likely), then send more $$$ to the candidates with the highest score, or perhaps those with the highest delta compared to previous period.

There is something wrong with that picture... The goal of our application is to identify where one *will* get the most bang for the buck, not where one *already* gets the most bang.

For example, let's say that the organization gathers data weekly, so that it knows the $$$ and outcomes at that periodicity. Two weeks ago, John Doe was sitting at 10% of the poll, while his opponent Jack Satan was at 60%. Last week, the organization spent $1,000 on John Doe's campaign and found that the poll numbers came out as Doe, 18% and Satan 50%. Is this sufficient information to drive a spending decision this week? The answer is "of course not", since the logical answer must weigh John Doe's chances against all the other 'Our Candidates' for the same body (say, the House of Representatives), as well as the other 'Our Candidates' for more significant seats (say, the Senate). Who to aid, who to abandon? Economists call this quandary "opportunity cost"; if I have the ice cream cone, I can't also have the fish and chips (easy decision, that).

The answer to the question is "triage of the candidates", and "data" is how one accomplishes this. For those that haven't watched "MASH" in a while, triage is the process for dealing with mass casualties, and given the number of elections conducted in large countries, mass casualties is an apt metaphor. So, the rules of triage:

1. Let those too sick to help, die.
2. Let those healthy enough to live, go on without intervention.
3. Help those who might live with ... help.

In the case of John Doe, depending on other factors he's a goner; sending limited money his way is wasted. By 'other factors', I mean such things as time left to election, or his opponent's closet full of skeletons soon to be revealed. How, on the other hand, do we identify the non-obvious Lazarus? Can data be of any help? If so, how to do this?

Is an RDBMS the solution?

And, of course, the answer is "mostly, no". "Mostly", because the answer is statistical and graphical in nature, about which the relational model has not much to offer analytically; these disciplines are designed around the cursed flat file. For operational purposes, one would expect that most of the expenditure data will be in some database. It's also safe to assume that this will be the driving data-store for the application. So, while we can expect that some of our data will be stored in a relational database, and that off-line data can be loaded with it, the relational capabilities don't solve our problem: How to find the best candidates to support.

Once we have extracted the predictions, the second objective is to find the optimum way to display the predictions. For this illustration, I'll say that they've decided on x-y plots with regression fitted. R offers an embarrassment of riches with regard to graphic display of data. You really should have a look.

In this example, I'll ignore the client side development, which could be Ruby on Rails, Cakewalk, some .NET framework, Struts, TurboGears, and so on; the whole point is to be client code agnostic (for an real world example, visit Xtuple). Right now, all I'll describe is using the statistics and graphing capabilities of R directly from a RDBMS, Postgres, in order to accomplish one task. I'll assume that the client side coders will decide where and when to display the analytical results. Those results, I'll assume for this illustration, will be some plot of our composite data, including a predictive component. The applied statistician often turns to linear regression for this situation. Econometricians and psychometricians (biostats, not so much) are long time users of regression analysis.

The implementation

We can assume, for this illustration, that we don't have access to the apparatchiks' proprietary data.

There is a trove of data at the FEC (Federal Election Commission), but I won't be using the FEC data here, since I want to build a database which directly addresses the triage problem with data that the organization should have. For those interested, one can play quite a lot with FEC data—an R developer has written an access routine in Ruby. Ruby is easily installed, and I'll assume that you'll do that if you want to explore the FEC data. Again, the point of this piece is to explore data for the purpose of finding the wounded who can be saved; not so much the technology, beyond R, which is quite to the point.

We proceed to winnow our data with some multivariate analysis. The essence of inferential statistics is correlation. I assume that the apparatchiks have many distinct data values available beyond what we can find in public sources. In order to demonstrate the process, I'll make up some data names and some data. In the real world, our apparatchiks will have (hopefully) more.

Triage: The Tech Bits

First, I found this link as I wrote today, which is further evidence of my thesis: analytics/stats will be done from the database. Whether these will only be various open source databases (and, considering Oracle's behaviour, MySql is no longer the most likely candidate) and stat packs remains to be seen. But the momentum is building. I am by no means a fan of NoSql type datastores, and for R support, it seems a bad match: That said, events move apace.

As I was preparing the TechBits, PostgreSQL v 9.1 went official. One of the new features is a **K nearest neighbor indexing**. Unless the Postgres folks enjoy confusing their users by munging a term they should be well aware of, **K nearest neighbor** is used in classification analysis, which R supports: More evidence that stats are being merged with the relational database.

For the purposes of this demonstration, I'll assume that, whatever the underlying catalog, a simple view can be constructed for our use. I'll also abstract away various niceties such as sql injection barriers (Postgres does it much the same as others), data normalization, and any other tasks needed to concoct an appropriate view.

Here's the data table:

```
CREATE TABLE "public"."events"  (
   "period"       int4 NOT NULL,
   "candidate"    varchar(25) NOT NULL,
   "event"        int4 NOT NULL,
   "amount"       int4 NOT NULL,
   "outcome"      int4 NOT NULL,
   "choice"       int4 NOT NULL,
   "guncontrol"   int4 NOT NULL
   );
```

The categorical columns, choice, guncontrol, are common 5 point scale from "Strongly Disagree" to "Strongly Agree", and reflect the candidate's position, not the voters (in this data, John Doe displays fungible positions). Adding the voters' view is certainly useful. I'll also assume that these are the minimal wedge issues, though more can be expected; which leads to the matter of data for stats. Stat calculations just don't work on character data, so it must be translated. Here, they have already been. The same translation is required for event, so I'll define the value/string such:

1. debate
2. canvas
3. rally
4. radio advert
5. TV advert

The ordering is approximately monotonic in cost to execute; the apparatchiks would order them that way, in any event.

The naive' notion is that measuring outcome, which in this data is the normalized polling percent, e.g. 11 is .11 or 11%, after spending money is sufficient information to shape further expenditures. The organization will have data for all candidates across the country; the question is who gets how much this week or month or quarter. Visual display of two variables is simple: a scatterplot with a simple regression line through the points. And, since in this notion, the data are nicely continuous, it makes sense in the arithmetic.

In Postgres, all "code" is stored as functions (functions, stored procedures, and triggers). Here's the function to generate a scatterplot and regression lines for our data.

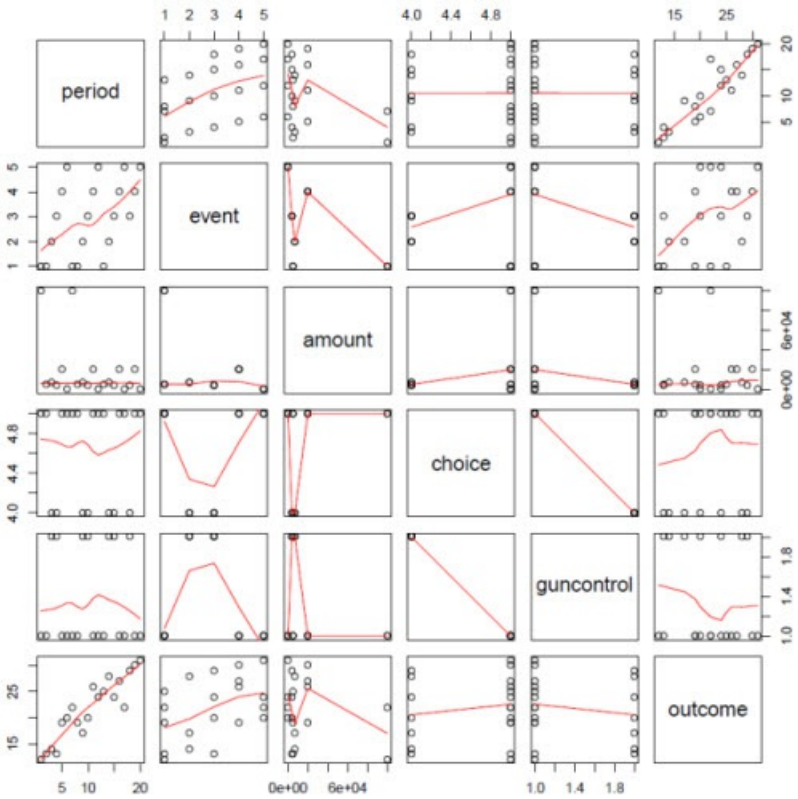
```
CREATE OR REPLACE FUNCTION "public"."pairs_graph" () RETURNS text AS
$BODY$
X11(display=':5');
events <- pg.spi.exec ("select period, event, amount, choice, guncontrol, outcome from
public.events where candidate = 'Doe' ");
pdf('pairs_graph.pdf');
pairs(events,panel=panel.smooth);
dev.off();
print('done');
$BODY$
LANGUAGE 'plr'
```

- The $BODY$ lines are markers to the engine for code blocks.
- The X11 line is required by Ubuntu to write graphs to files; the reasons are covered in the Ubuntu link.

- The events line does the database work; the <- is assignment to a dataframe, what we want as the explained variable, outcome, is listed last for visual ease.
- The pdf line tells R/Postgres where to write the graph; this will be in Postgres' home path.
- The pairs line does the work; it generates pairwise scatterplots and fitted regressions (many options are available).
- The dev.off line disconnects from X11
- Finally, print sends a soothing message to the console.

Note that language is called as 'plr'; it's actually C, since R (entry points, in any case) is a C application.

We live in a three dimensional world, cosmologists not withstanding, and even displaying three axes of data leaves most people puzzled. We want to predict outcomes driven by more than one predictor, money. We want to do it graphically, but we really only have two dimensions to work with. It turns out that R provides support for that. Traditionally, multiple regression would print text, not graphs.



What does this tell us? The row we care most about is the last, where outcome is plotted against each of the explanatory variables. Time is a great healer in this data. Amount spent, not so much. Most significantly, R fits what are called localized regression lines which adapt the slope of the fit for changes in the scatter; this is why the lines aren't just straight, as one would get from a simple regression. The advantage of this method is better fit, at the cost of complexity. Localized regression is a stat algorithm, not specific to R.

Interpretation during such a PoC/prototyping/agile exercise raises meaningful questions for our apparatchiks. Here are just a few.

Do we find, looking at the other rows in the panel, that our presumed independent variables are correlated? The real world implication, if there is correlation, is that overall predictability of our variables isn't quite as good as the calculations show (we don't print them for this GUI widget). It also could mean that these factors' correlation was unknown to us. We've learned something about our data; some variables affect others in ways we didn't understand. For example, gun control and choice are negatively correlated. That's likely true with real data, too. Depending on the attributes tracked, correlations among the explanatory variables offers the opportunity to hone our funding process. There may be insurmountable effects in the electorate such as gun control and choice. There may be opportunities; immigration may be less significant than hypothesized.

Do we measure outcome with just one source? Sources can include our pollster who may gild the lily a bit to keep the Suits happy, Gallup, or similar, polling, name recognition surveys, social network mining (what happens on twits/tweets and facebooks, etc. after an event), radio/TV ratings, opposition response. We can define a composite outcome, weighting each component as the apparatchiks see fit.

How do we measure outcome, point in time or delta? We can see the trend on the period graph in the panel using point in time. We could also plot delta-outcome against the variables. It may well be that $10,000 will be enough to move from 10% to 11% (a 10% gain), but a 10% gain on a base approval of 40% (delta-outcome 4 points) may never happen; the curve bends down.

What is the weight of John Doe in the organization's nationwide plan of attack? Does Doe matter more or less than Smith in another district? Is Doe a shoe-in? All of these matters impinge on the funding decision which the stats are intended to guide; who among the wounded are most likely to prevail with assistance? Who is gaining traction, and who is slip sliding away? We can now query our database, and provide a semblance of an answer to our suits in a pretty GUI.

Since we can graph for an individual, do we also provide graphs on aggregates; for example, all candidates for all seats in a state, or by region

(here in the States, Red versus Blue is popular)? We can aggregate up in either sql or R. A reasonable UI would be a map of the country. A right click on a state/province/canton would bring up a list of candidates by district, the user picks the candidate then chooses the factor attributes to plot (in addition to outcome, event, and amount), and choosing would return the panel for that candidate. A rollup at the state/etc. level would need weights for each candidate, assigned by our apparatchiks; upper chamber being more important, influential districts, and the like.

This panel display, with half a dozen variables, is about the limit for a naive' (from the point of view of stat sophistication) user. With exposure, and perhaps some instruction, sophistication develops rather quickly and the response is always for more data. At one time, I taught one week stat courses for the US government. We didn't have R, but the kernel of stat thinking gets through. That kernel: from a computational standpoint, it's all about minimizing squared differences. We tell R to find the best model to fit the data we have (and *not* the other way round), by minimizing the differences between the data points and the model's graph. There's a whole lot of maths behind that, but you only need to know them if you're determined to acquire a degree in stats. Or write R routines! What applied stats (by whatever title) folks need to keep in mind are the assumptions underlying the arithmetic their stat pack is implementing. Most often, the assumption that matters is normality in the data. The stats meaning of "normal" is similar to what RDBMS folks mean. Both fields have orthogonality as a principle: for RDBMS, "one fact, one place, one time"; for stats, all independent variables are also independent of themselves. The larger meaning for stats folks is that the variables follow a normal/bell curve/Gaussian distribution. This assumption should be confirmed. Stat packs, R included, provide tests for this and should be exercised and methods for transforming variables such that they are more nearly normal in distribution.

But for our apparatchiks, there's plenty of exploration to be had. That would be done in the usual fashion: connecting to a relational database from R in R Studio. From that vantage, correlations in the data, hints of what other data might be useful, what sorts of visual displays, and much more can be discovered. At that point, creating functions in Postgres to present the visuals is a small matter, the script in R Studio can (nearly) be dropped into the Postgres function skeleton, just add the SQL query.

As above, the client side code will have some glyph to click, which represents a candidate. Clicking the glyph will bring up the plot. For a web application, we need only have the image file on the web server's path. We'll supply some form of a graphics file such as png or pdf. For my purposes, providing a point plot in an OS file will suffice. In fact, since all industrial strength databases support BLOBs/CLOBs, we could store images in the database, and serve them up (I'll leave that as an exercise for the reader). (For what it's worth, my research showed that discussions come down on the side of OS files rather than BLOBs.) For a web application, writing to an OS file is fine; we only need to fill in the img tag to get our plot.

In order to run R functions, we need some help. That help is in the form of PL/R, the driver that "embeds" the R interpreter in Postgres. Once PL/R has been installed, we can use R functions in our SELECT statements. We can then run R functions in Postgres that produce text output, and, more to the point, graph image files. These can then be shipped off to the client code. Kinda cool.

So what does the future hold?

For some time, ANSI SQL has been accruing ever more complex statistical functions; there's even support for regression, although I must admit, I've never quite understood why—anyone experienced enough to understand what regression does wouldn't trust a RDBMS engine (well, its coders, really) to do it justice.

When IBM bought up SPSS Inc, it sent a shiver down my spine. A good shiver, actually, since I'd been born into professional employment as an econometrician, and used SPSS in its early mainframe clothes. I expect DB2 to include SPSS commands, thus thumbing their nose at ANSI. But, with other developments in the hardware world (multi-core/processor and SSD) why not? What with some ANSI sanctioned stat calculations, the BI world nudging up against the OLTP world (near real-time updating), and the advent of (possibly) full SSD storage for RDBMS, what might the near future look like?

Now that IBM has SPSS, and Oracle has bulked up on some stats (but I don't find a record of them buying up a stat pack vendor, and what they show is nowhere near R), might SQL Server integrate to R? Or, Oracle? It'd be easier for Oracle, since the engine already supports external C functions. SQL Server is mostly(?) C, so adding C as an external function language shouldn't be a mountain to climb.

R is GPL, and calling GPL software doesn't make the caller GPL; that's been established for years. If it were otherwise, no-one would run Apache or most versions of WebSphere. Calling into R from the engine is no different. The PL/R integration was written by one person, Joe Conway, and given that both Oracle and Microsoft have legions of coders at their disposal it wouldn't be much more than a molehill for them to write their own integration. And if the Oracle or Microsoft coders chose to write a new solver for R, that code would also be under GPL.

My hope, and I'll qualify that as a 'guess', is that "fast join databases" will become the norm (since they'll be highly normal), which, along with federation, removes the need for separate DW databases being fed from OLTP databases. There'll just be one database, just like Dr. Codd envisioned, supporting all applications in need of the data. And, yes, decent federation support deals with disparate data types across engines. With stat pack syntax built into SQL (SSQL, perhaps?), the database server becomes a one-stop shopping center. Just don't expect an ANSI anointed standard.

Bibliography and links

I've ordered the bibliography "from soup to nuts", rather than the traditional alpha sort. The texts are those I have used, and a few with high regard in the community that I've not gotten to yet.

- Postgres installation.
- Matthew and Stones: "Beginning Databases with PostgreSQL". Almost everything needed to get running with Postgres.
- An R installation instruction.

- R for Windows: FAQ.
- R Studio. An open source, all platform, IDE entrance to R. Quite new. To quote the end of Raiders of the Lost Ark, "It's beautiful". No, your head won't melt.
- Installing PL/R. From the Onlie Beggeter of PL/R; follow the link back.
- Using PL/R. Very helpful information.
- Ruby: If you only want this for FECHell, follow the links to 1.8.7; 1.9.x and FECHell don't get along. Otherwise, use rvm or similar to have multiple Rubys on your machine.
- Python. Take 2.7.x; Python is the lingua franca of the R/stats world. Optional to this article.
- FECHell. If you want to explore US election spending...
- Adler: "R in a Nutshell". The most comprehensive, although necessarily not deep, coverage; if you've got a question, you're most likely to find the answer here.
- Maindonald and Braun: "Data Analysis and Graphics Using R". I have the second edition, from 2007; from the comments, there's not much difference. Stresses the graphics, and the R code is relegated to footnotes!
- Crawley: "The R Book". The best all rounder.
- Chambers: "Software for Data Analysis". Written from a programmer's, rather than analyst's or user's, perspective.
- Wickham: "ggplot2 Elegant Graphics for Data Analysis". The latest plotting function, in depth. The graphs and plots here weren't made with ggplot2, I've not dug into it enough.
- Venables and Ripley: "Modern Applied Statistics with S". The lodestone text; syntax is S, and from 2002 or thereabouts.
- Cleveland: "The Elements of Graphing Data". Cited, where any book is, as the graphics source in R; if you were expecting Tufte (I was), sorry.
- Hoel: "Introduction to Mathematical Statistics" My stat book.
- Janert: "Data Analysis with Open Source Tools" A practitioner's take on data analysis, not mostly R and not mostly regression; likely an older guy, with much snark on offer, and much of it earned.
- Ubuntu specific gnarls: Some needed tidbits dealing with plot production.

This is my fave online tutorial, although I've not got the book derived from the tutorial.

And, for all things R online, there's R-bloggers.

Glossary

- **BI**: Business Intelligence
- **BNCF**: Boyce-Codd Normal Form
- **BLOB**: Binary Large OBject
- **CLOB**: Character Large OBject
- **DBI**: Database Interface
- **DDL**: Data Definition Language
- **DRI**: Declarative Referential Integrity
- **DW**: Data Warehouse
- **FEC**: Federal Election Commission
- **GPL**: General Public License
- **OLTP**: OnLine Transaction Processing
- **OS**: Operating System
- **PL/R**: Procedural Language for R
- **RBAR**: Row by Agonizing Row
- **RDBMS**: Relational Database Management System
- **RJDBC**: a package implementing DBI in R on the basis of JDBC (Java Database Connectivity)
- **RM**: Relational Model
- **RODBC**: a package implementing DBI in R on the basis of ODBC (Open Database Connectivity)
- **SAS**: Statistical Analysis System
- **SPSS**: Statistical Package for the Social Sciences
- **SSD**: Solid State Drive

Pandas: below "C-level", and drowning

Published Monday, September 26, 2011 4:50 PM

Last week, at the Technical Communication UK conference, I did a little lightning talk. It wasn't very nice. This is perhaps unsurprising, given that the session was billed as "rants" - an opportunity for people to get things off their chests. Boy, did I.

In fact, my first line was more or less:

> *"You're sleepwalking into obsolescence at the hands of skinny-jeaned Hoxton web twats."*

It got worse from there, but I do feel it merits some clarification, and a smidge less precocious grandstanding.

You see, cheap ad-hominem slights against one half of my professional community aside, the other half does risk being made obsolete by them. Although there are all sorts of reasons why that might be OK, or even better than OK, it's also potentially worrying.

My rant was spurred by attending the Content Strategy forum. It was a great event, but a couple of things struck me. One was that almost nobody mentioned technical communication. This is odd, because technical communication has been the unwitting (and certainly un-labelled) home of the practices of content strategy for years beyond number.

The other was that - despite this - most of the examples and case studies were tech comms projects. There were several examples of help system re-designs, and of noodling about with FAQs. Des Traynor talked about user interface text; Ove Dalen told us about Tenenor's sales-boosting, cost-saving web support redesign; and Gerry McGovern got a round of applause for the worthy sound-bite "support is the new marketing".

But they weren't talking about "tech comms" or even "documentation". It was "support content", and it had a big, sexy ROI.

So where were the tech authors?

Despite bearing the brunt of my opening slur, the rant wasn't laying this at the door of the content strategy community. That community's quite focused on consultants and agencies, and those consultants and agencies have been really rather successful at selling themselves to organizations. They've had that success by offering good solutions to hard problems, and they came to the CS Forum to talk about it. Good on them.

But where were the tech authors?

Here's the end of my rant:

> *If your organization is hiring a content strategy consultant to deal with your "support content", isn't it just possible that your tech comms department might have slipped up?*

If content strategy is making money in this space, has tech comms failed?

The answer, of course is "no, but..." and there are a world of caveats, not least that plenty of organizations still don't have technical communication departments. But many of those that have them also have a problem. I would argue that the problem is visibility and influence.

You see, a great deal of content strategy has its roots in marketing, and a great deal of content strategists are used to making a strong ROI case. Lots of them are used to doing it to senior people, too. Content strategy is just rather better groomed for making a case at what is loathsomely referred to as "C-level". That is: to your CEO, or CMO, or even CIO; to the folks in the big chairs who worry about the bottom line. Whereas, historically, tech comms is somewhat more habituated to cowering in the basement, bashing out a 400-page pdf and hoping to escape the next round of redundancies.

We need to fix that, right?

Or not. I mean, provided the tech comms work is getting done right, I'm not sure it matters who does it, or under what job title, and I've no intention of starting a tech comms/content strategy turf war.

But there is something else here. Something a bit fuzzier and fluffier, and it has to do with agency and happiness. You see, when content strategy has a voice on the board, where it has buy-in and investment, it can get stuff done. Life's a lot less grinding and miserable in that world. Far fewer people breathe down your neck when you're a revenue centre as opposed to a cost.

So even assuming we don't want to get our egos out and see whose is bigger, tech comms still needs to get its act together. It still needs to talk to the board. It needs to demonstrate its value. It needs to not march peacefully into obsolescence.

That's where the pandas come in. They're lovely, aren't they? Large, cute, and somehow reassuring. Only, there's a common joke about pandas. You've probably heard it: does a species that's too lazy to mate really deserve saving?

I won't invite you to consider technical authors mating; this isn't about that. But it's worth pondering just who will feel sorry for tech comms if it goes

down quietly. And as a means of not doing so, it's worth rather more to actually talk to the content strategy community. They're doing the same job. I hope they're doing it as well, and I *know* they're better at selling it.

On a related note - I'm running a survey on who looks after user interface text. It's very short, and you can fill it in here.

by Roger Hart
Filed Under: Content strategy, Technical communications, angry ranting

Jez Humble: Geek of the Week

28 September 2011
by Richard Morris

Jez Humble and David Farley achieved fame through a book that tackled the least glamorous but most intricate part of the application development cycle, Deployment. It was no accident that the book achived so much attention, since it was a lively and iconoclastic take on a vital but neglected aspect of development upon which the ultimate success of software projects so often depend. We found Jez to be an interesting guy, too!

Jez Humble immersed himself in technology at the start of the Millennium, the year the dot com bubble burst, the time when the masses stopped believing there was a new world upon us and that the hype didn't live up to its promises.

Nevertheless, his interest in computers survived and in the eleven years since has worked as a developer, system administrator, trainer, consultant, manager, and speaker. He has worked with a variety of platforms and technologies, consulting for non-profits, telecoms, financial services, and online retail companies.

He has worked for ThoughtWorks, the pioneers of Agile, in bases around the world including Beijing, Bangalore, London, and San Francisco.

He co-wrote 'Continuous Delivery - Reliable Software Releases Through Build, Test, and Deployment Automation' along with David Farley. This was published by Addison-Wesley in Martin Fowler's Signature Series. This book reinforced the theory that good software can be achieved by following a series of managed approaches which are carefully planned and rigorously implemented. The book has been influential in focusing attention on one of the more neglected aspects of the development cycle, the delivery and deployment process; the meticulous steps required to ensure the successful creation or update of a working production system. Although the book concentrates on being a general guide to successful software delivery in the business environment, it provides a strong advocacy of both collaborative working methods between testers, developers and Operations, and automation of the processes. It was an ambitious book that couldn't always offer solutions, and whilst it sometimes caused controversy, it has quickly become adopted as a standard text for Agile. From both experience and research, it describes the principles and technical practices that enable rapid, incremental delivery of high quality applications.

In his spare time, he writes and reads music and books. As well as his earlier degree, Jez has masters in ethnomusicology from the School of Oriental and African Studies, University of London, for which he studied Hindi and Indian classical music. He's married to Rani and they have a baby daughter.

RM: You studied physics and philosophy at Oxford. When did your interest in computers start and how did that develop into a career?

JH: I became interested in computers when my neighbours got a ZX Spectrum. I got one for my 11th birthday, and I was hooked. My favourite computers were Acorns. My parents bought me a BBC Master and then an Archimedes A3000, which were brilliant because they had really nice programming environments (for the time) and great reference manuals. I stopped being interested in computers when I nearly got kicked out of school for illicitly acquiring administrative privileges over the school network, which was coincidentally also around the time the school started admitting girls. I got back into computers after I graduated from university, because I needed to make a living.

RM: Staying with the philosophical questioning for a moment what makes computer science an empirical science and not merely a branch of pure mathematics?

JH: The extent to which mathematics tells us anything useful about the real world – and can thus be considered empirical – is still hotly debated (Einstein once said "as far as the laws of mathematics refer to reality, they are not certain; and as far as they are certain, they do not refer to reality.") Fundamentally the output of mathematics is created through symbol manipulation, which is a purely intellectual exercise. If you're doing science, those symbols have to correspond to measurable things in the real world so you can do experiments to see if your models have predictive power. Even complexity theory is arguably empirical science rather than maths inasmuch as it purports to tell us about the real-world properties of algorithms (the amount of time they take to execute, for example).

In terms of the activity of programming, once you move beyond the narrow context where formal methods are appropriate, I think test-driven development is a really nice way to expose the fundamentally scientific nature of writing software. First I write a test that makes a prediction about the behaviour of the method or function I'm about to write. The test fails, because I haven't implemented the function yet. Then I write the code that makes the test pass. That's science in its purest Popperian form. Of course programming is an art too, because you have to come up with the correct tests to write, and the right order to write them in, if you want to create an elegant and valuable program.

RM: Do you think we'll get to the point when machines can be made to think?

JH: I reject the Cartesian view that there is some kind of thinking "stuff" that is ontologically distinct from matter, so I don't think there's any fundamental reason why machines shouldn't think. However I think it's a while before we'll get there. Some of the bits of philosophy that I continue to find fascinating (and insufficiently examined by computer scientists) are Nietzsche, Heidegger, and Wittgenstein's views on consciousness, being, and language. Ultimately the ability to "think" comes from the combination of some innate mechanics that undergoes a process of socialization. Historically we've focussed a lot of effort on the innate mechanics, and not so much on the process of socialization. That probably says more about computer scientists than it says about the nature of the problem.

RM: In one of his papers Dijkstra talks about how computer science is a branch of mathematics and how computer science students shouldn't touch a computer for the first few years of their education and should learn instead to manipulate systems of formal symbols. How much mathematics do you think is required to be a competent programmer?

JH: I can only speak – as I suspect Dijkstra did – from personal experience. I learned to program computers several years before I studied set theory and formal logic. I won't deny that learning those things helped my programming, but I think that studying philosophy (and more practically, electronics) helped my programming as well. I think the most important things required to be a competent programmer are a desire always to be learning new things, combined with a masochistic tendency towards delayed gratification that manifests itself in problem solving (I'm thinking here of the hours I spent as a teenager debugging ARM assembler printed out on fanfold paper while my classmates were out getting drunk). You probably need similar personality traits to be a competent mathematician.

RM: For people coming into the industry, are there other skills, beyond the ability to write code, that they should develop?

JH: I think that if you want to be a great programmer, you need to be a generalist. You need to have at least a high level understanding of systems administration, CPU architecture, networking protocols, and so forth simply because all the abstractions we have developed to make software development faster and more efficient are leaky. Knowing when the context in which you can ignore what's going on under the abstraction layer you're using at any point in time (even if that abstraction layer is assembler) is as important as making sure that the program satisfies its functional requirements.

It's also important to think holistically (this is sometimes called systems thinking). Code doesn't deliver any value until it's in the hands of users – indeed you can't know whether or not it's valuable until you get feedback from users. Until then, any work you're doing is based purely upon a hypothesis. Understanding what is valuable to your users – which means understanding something about their world – and how to best deliver that value requires much more than just knowing how to write code. Specification documents, for example, are a very leaky abstraction.

RM: Has programming become a more social activity than it used to be? Do you remember any particular 'aha!' moments when you noticed the difference between working on something by yourself and working on a team?

JH: Writing valuable, high quality software is fundamentally a social activity, if only to the extent that judgements of value and quality are subjective. Too many programmers fail to create valuable software because it's so easy to focus on the immediate issue – getting some particular part of a program to work, or creating an elegant architecture – at the expense of actually solving a problem. It requires discipline to constantly ask oneself, "is what I'm doing at the moment actually the most valuable thing I could be doing right now?" Other people help keep you honest.

My biggest "aha!" moment came when I joined ThoughtWorks and got to do pair programming and test-driven development with people who were really great programmers. I had been pretty sceptical about the whole XP thing, but I learned more about how to create, deliver, and evolve high quality object-oriented software in my first year at ThoughtWorks than I'd learned in all the years before that. However I try not to be bigoted about pair programming and TDD because I don't believe there's any way to convince you they are a good idea unless you've actually experienced it for yourself. They're neither necessary nor sufficient conditions for creating and evolving great software, but they make it much easier and much more fun.

RM: Though it's not new, Continuous Delivery, the title of the book you wrote with David Farley and which was published in 2010, made the concept much more popular. Roughly speaking it means that every build goes through the same quality process, so it saves time, reduces risk, improves trust and helps deliver better software to clients.

To look at this laterally, I suppose academics have worked this way for years. Are there areas where industry is ignoring good stuff about how we should build software? Probably the best example I can think of is where Intel had to recall the Series 6 chipsets because of a serious bug, something which cost an estimated $600 million.

JH: One of the problems in our industry is the divide between academia and commercial software development. It's hard to move between the two because there is a tendency for each branch to consider itself somewhat superior to the other (you see the same attitude in physics between the theoreticians and the experimentalists). So I'm sure there's plenty of stuff the two sides could learn from each other.

A great example comes from my co-author, Dave Farley. He works on the world's highest performance financial exchange, LMAX. The main thing they did to get really blistering performance was to throw away the conventional wisdom on how you create these kinds of systems. So, to pick two particularly counter-intuitive decisions, their core computational engine (which they recently open sourced) is single-threaded and written in Java. They spent a lot of time thinking about how to write algorithms that displayed mechanical sympathy with the JVM and the underlying CPU architecture. Their approach is a striking example of how you can make a giant leap when you harness together theoretical and commercial approaches to writing software.

RM: Do you think programming and building software should be like an engineering discipline? The analogy of building bridges

comes to mind because you can predict how it is going to take and the bridges for the most part don't fall down?

JH: The great thing about bridges is that if you're building (say) a truss bridge, there is a nice model where you can plug in some variables and out pops the design. A software system of any size and complexity is going to be way too complex to model analytically (which is why formal methods in computer science aren't widely used in real life). Even when we build things where there is an applicable model that is tractable analytically or that can be simulated on a computer things can go wrong, as the new terminal at Charles de Gaulle showed. But building large software systems is not really comparable to building a bridge.

My wife and I visited Gaudí's la Sagrada Família in Barcelona recently, and one of the striking things is the extent to which the design and the construction of it were performed iteratively and incrementally. Gaudí built a smaller church using his new hyperbolic approach before he started work on the Sagrada Família. He had a big workshop in the crypt where he was forever creating scale models to test out his ideas. One of his innovations was to create upside-down models of parts of the building with suspended weights simulating the loads on the structure to validate that it was sound (see image below).

Photo Credit: Subtle_Devices' Flickr stream

Now, the Sagrada Família has been under construction for over 130 years. It is behind schedule, way over budget, and still not done, so in that sense it's perhaps not a shining beacon of what we should be aiming for (Gaudí once remarked "My client is not in a hurry")." First, it's much cheaper to experiment. Second, if we approach the design correctly, we can start performing useful work with computer systems from early on in their lifecycle, and get feedback on what to build next so we don't waste time building stuff that people actually don't want. That's why the concept of a minimum viable product is important, and where continuous delivery comes into the picture.

RM: When it was released in 2010 your book was recognized as one of the most important software development books of the year. What books would you recommend up and coming programmers to read? What technology books do you read? Have you read Don Knuth's Art of Computer Programming for instance?

JH: I bought The Art of Computer Programming in India when I was working in ThoughtWorks' Bangalore office and got through most of Volume 1 on a road trip, but I never found the time to finish it. I still dip into it from time to time, because occasionally it feels good to mainline pure computer science from the motherlode.

Among the books that I've read, I think probably my favourites are Bob Martin's Agile Software Development, Michael Nygard's Release It!, Nat Pryce and Steve Freeman's Growing Object Oriented Software, Guided by Tests, and then Don Reinertsen's Principles of Product Development Flow for the process wonks.

I've read enough of Knuth's opus to know that it wouldn't be a good use of my time to finish the rest of it any time soon. That's not to disparage it – it's clearly a very deep and important piece of work – it's just not that relevant to what I'm doing at the moment. Among the books that I've read, I think probably my favourites are Bob Martin's Agile Software Development, Michael Nygard's Release It!, Nat Pryce and Steve Freeman's Growing Object Oriented Software, Guided by Tests, and then Don Reinertsen's Principles of Product Development Flow for the process wonks.

RM: Do you think programming, and therefore the kind of people who can succeed as programmers, has changed? Can you be a great programmer operating at a certain level without ever learning assembly or C?

JH: Certainly there have been serious productivity gains in programming over the last few decades because of the new tools

we have available (I am using the term "tool" pretty widely here, to include paradigms such as object-oriented and functional programming). But as I say I don't think you could really be a great programmer without having at least some understanding and sympathy for what's going on under the hood.

RM: I guess that the most important rule of technical writing is to understand your audience – the better you know your reader the better you can write. What would you say are the other cardinal rules? Is there a way of saying everything twice in a complementary way so that the person who's reading it has a chance to put the ideas into their head in a way that reinforce each other?

JH: Everyone learns differently, so it's hard to be general. There are a bunch of different ways of organizing your content. Some people like a more discursive style. Some people prefer a terse style where things are laid out more formally (check out Spinoza's Ethics for an extreme example of this – maybe there's someone, somewhere for whom this book represents the acme of technical writing). Martin Fowler has pioneered a two-part model where you introduce the core concepts discursively and then the rest of the book is patterns.

Continuous Delivery is written in a discursive style throughout where we deliberately trade off being concise both for readability and so people can dip in to a particular section and get all the context they need to understand it. Some people don't like it for this reason, and it certainly made the book longer and more irritating if you read it cover to cover, but we tried to achieve the goal you describe of showing how everything fits together in a holistic fashion, and it's very hard to achieve that without some level of repetition.

RM: It's often claimed that there are orders of magnitude differences in productivity between programmers. I read an article which debunked these claims saying that studies found this were done some time ago and a lot of things have changed about programming and working since then that could have accounted for the differences such as some in the study were using batch processing techniques while others were using timeshare programming environments. What's your view on this and how would Lean and Agile methods help with productivity?

JH: From personal experience, I can tell you that it's true that there are orders of magnitude differences in productivity between programmers. Some of that is down to familiarity with their environment. Some of it is down to having a wide variety of experience. Some of it is down to being very clever.

I think that lean and agile methods improve productivity by setting up fast, rich feedback loops so you can work out whether what you're doing at any point in time is valuable and high quality as fast as possible. The biggest source of waste in software is functionality that is developed but never used. The Standish Group presented a report at the IEEE XP 2002 conference, based on studies including data gathered at the Department of Defence that showed that over 50% of functionality developed is rarely or never used.

Agile and lean methodologies help eliminate that by trying to get the ultimate feedback loop – from users back to the team – as fast as possible, and also by setting up other feedback loops through techniques such as continuous integration and test-driven development so you can find out quickly if you've introduced a regression or if what you've written doesn't actually behave in the way you're expecting.

However none of this can compensate for having mediocre developers. Having 10 really good programmers is always going to get you better software faster than having 100 mediocre ones.

RM: What is your process for designing software? Do you fire up Emacs and start writing code or do you use UML as a design tool technology or just start coding?

JH: These days when I fire up Emacs it's to write books rather than code, but I have to say I've never used UML formally. I would hazard that even Martin Fowler, whose best-selling book is UML Distilled, would not advocate the model-driven approach to design.

Predictably, I advocate just enough design. Based on the limited information you have before you start writing any code, grab a whiteboard with a bunch of people on your team, and discuss the various options. Come to a decision – without spending too long arguing over minutiae – on what you think is going to be a good possible approach. One of the attributes of a good approach is that it lets you test the functional and cross-functional (performance, security and so forth) characteristics of your system from early on in its lifecycle, and that you can change it without too much trouble if it turns out not to be so good. As necessary, write some throwaway code to test your more risky hypotheses.

Then as you write the real code, write automated tests that assert the behaviour that is important to you so you can validate your architecture as soon as possible when it's cheaper to change it. Always make decisions based on data, not intuition. Intuition is useful in software development, but computer systems are inherently complex and non-linear, and intuition must always be validated with real data. To quote Knuth, "Premature optimization is the root of all evil".

Presentations: On snatching victory from the jaws of defeat.

Published Thursday, September 29, 2011 3:59 AM

Ah, SQL Pass is imminent; and all over the world, people are preparing their presentations. There is great satisfaction to be had through facing a specialist audience. For the rest of the year, even devoted friends and family tire quickly of hearing your breathless account of the inner workings of execution plans and distribution statistics; but here the audience hang on your every word, if you've pitched it right. Mind you, getting it wrong carries with it a great deal of pain. As you hold forth strongly on what you imagine to be a really hot subject, such as virtualization, suddenly hundreds of faces are illuminated in front of you as laptops are opened by bored members of the audience, who start checking their mail or updating their Facebook pages. You struggle on, and they start Twittering, damn them. I remember more than one occasion where the presenter, stung by the twittered criticisms, twittered back defiantly. Admirable spirit, I reckon.

There are, of course, occasions when the audience has some justification for feeling a bit cheated. One famous SQL expert accidentally left his mobile phone on and had a family member ring him half way through. In his panic, he answered it without backing away from the microphone, saying, "Not now, dear, I'm doing a presentation at SQLPass". This is a line I have cherished ever since. A tense conversation ensued, and the audience was agog to hear both sides. Eventually, the caller was soothed enough to hang up and we carried on with the presentation. However, five minutes later, the phone rang again and was again answered! .

Actually, the presentation was made more memorable by the intermission, and was subsequently listened to intently by the audience. I must admit I thought it was a gag and, afterwards, I put it to the presenter that what we'd witnessed was a trick to engage the audience and induce them to turn off their laptops and listen. Nobody forgets to switch their phone off twice surely? Truth, he assured me, was stranger than fiction; it was a brand new phone, bought that morning as a replacement for a broken one. He hadn't had time to learn which button did what..

All sorts of things can go wrong in a presentation; technology grenades, power gets cut, fire sprinklers go off, members of the audience get taken ill, police raid (wrong address). The seasoned presenter will be prepared for almost anything, with strategies for coping with any predictable turn of events..

When thinking of presentations that can go wrong, I'm always reminded of the time that the great poet and author, Hugh Iztu, got invited to China for a university symposium that included a celebration of his creative work. Naturally, I've disguised or changed the details, and it isn't his real name..

Hugh is a celebrated writer, one of the last of the 'beat' poets. He'd led an innocent life, gradually descending from those heady, reckless days of post-war rebellion into a cosseted life as a lecturer in a leafy Colorado university, lauded by intense-looking students wearing berets and dark glasses. He was able to savor those wonderful years when he hung out with Dylan and Joan Baez, and went carousing with Elvis Presley. His first books were a magical invocation of an angst-fuelled adolescence, a discovery of both rock and roll and sex, and the giddy cultural revolution of sixties San Francisco. Now a plump and dignified academic, with a string of acclaimed books that never really made it to the mainstream, he was surprised to receive an invitation to speak in China about his life and work, as part of a symposium about American art and culture. It was a vast venue, and the fees they wanted to pay seemed astronomical.

When the plane landed, a large number of people met him. Some of them could speak perfect English, and knew his writings far better than he did. All of them were keen to shake his hand and express how privileged they were to meet him. For the first time in his life, he felt he was a real celebrity. He was given a tour of the university and was amazed to find how rigorously the cultural development of post-war western culture, and its many twists and turns, had been studied, documented and understood.

After a day of meetings with editors, film directors, intellectuals, and well-known Chinese artists and authors, he left quietly and was taken to his luxury hotel. The following morning, all was quiet. His hosts assumed that he wished to meditate before the headline talk he would give at the city's stadium in the afternoon.

As is usual in modern international hotels, the blues quickly descend. Left alone, Hugh was suddenly struck by the immensity of the task before him; to engage that vast audience, and to transmit to them the emotional intensity of the beat era, the camaraderie with the Nashville set, and his subsequent elevation into a literary icon of the Sixties West-coast.

At that point he spied the drinks cabinet and remembered his host's cheery invitation to help himself. He did, and each glass increased his confidence. When the car arrived to take him to the stadium, he was surprised by the difficulty he had in negotiating the stairs. Inside his brain, the lights were slowly going out. The only bright glow was his alcohol-fuelled confidence.

At the packed venue, a huge screen flashed photos of him, young and fit, writing lyrics with Dylan, or speaking with sixties gurus. Even younger versions of himself appeared with Mae Boren, Col Parker and Elvis, or doing public readings in a baggy jersey with Ginsburg and Ferlinghetti.

As he shook hands and exchanged greetings with a queue of dignitaries, as he was guided towards the stage, the whiskey finally began to anesthetize the cerebral cortex, and Hugh was dimly aware that he had perhaps been unwise; he had almost lost the capacity to string two meaningful words together. An intellectual retrospective, a broad account of a lifetime's poetic work, a glimpse of the artist at work; these were all now out of the question. With his last lucid moments, he started to grasp about in his memory, like a drowning man reaching vainly for a lifebelt, and then, finally, grasped something solid from the recesses of his memory.

The audience saw Hugh's small figure appear on the distant stage, suddenly magnified by the huge screens behind him and leaned forward expectantly to hear his words. Here, they thought, was something they'd be able to tell their grandchildren in years to come. Hugh went up to the microphone, swayed uncertainly and blinked at the vast sea of heads in front of him. From some entrenched resource, deep in the cerebellum, Hugh wrenched his Elvis impression, one that had entertained generations of students.

"Well, since my baby left me.
I found a newplace to dwell
It's down at the end of lonely street
At heartbreak hotelâ€¦"

It is a useful fact about the human brain that, long after the capacity for logical thought is neutralized by alcohol, one can still do impressions. So engrossed had he become that he had no thought but to make it good. As he did it, the energy and excitement of his time hanging out with the Memphis beats and Rock artists came flooding back. The audience was stunned. They gaped at the huge figure on the screens as he pitched into the song. They were expecting a rather boring lecture full of worthy thoughts about literature and poetry. Instead, one star of western culture had somehow morphed into another, and here was a brilliant poet who had transcended language, with a piece of theatre that expressed, in an immediate way, that which mere language could not; the broad sweep of post-war western culture that had been denied to them for so long.

According to the account I heard, no visiting celebrity had ever received applause like Hugh received when the adrenaline finally ran out and he slumped semi-conscious into his seat. Tears of emotion ran down the cheeks of even the coldest intellectuals. It had tapped straight into something deep in the psyche of the newly-liberated intellectual life of China.

The moral? With public speaking, whatever the emergency, there is always a way of pulling splendid victory from the jaws of defeat, but you may, like Hugh, need to dig deep to find it.

by Phil Factor

A Performance Troubleshooting Methodology for SQL Server

28 September 2011
by Jonathan Kehayias

> When healing a sick SQL Server, you must forget the idea that there could ever be a simple correspondence between symptom and disease: The art of troubleshooting is much more the art of discovering, and assembling, the various pieces of the puzzle so that you have a complete understanding of what is going on inside of a server

Knowing where to start is the toughest part of solving a problem. As a Senior Database Administrator, I prided myself on being able to pinpoint the root cause of problems in my servers, and quickly restore services to normal working order. The ability to do this is partly down to a sound knowledge of your SQL Server environment, partly to having the right tools and scripts, and partly to what you learn to look out for, based on hard-earned lessons of the past.

Nailing down a specific methodology for troubleshooting problems with SQL Server is hard because, of course, the exact route taken to solve the problem will depend on the specific nature of the problem and the environment. One of the keys to accurate troubleshooting is not only collecting and examining all of the relevant pieces of information, but also working out what they are telling you, collectively. There is a famous old proverb, recorded in John Heywood's *Dialogue Containing the Number in Effect of All the Proverbs in the English Tongue*, which sums this up very well:

I see, yet I cannot see the wood for the trees.

If you collect and examine individually five separate pieces of performance data, it's possible that each could send you down a separate path. Viewed as a group, they will likely lead you down the sixth, and correct, path to resolving the issue. If there is one take-away from this article, it should be that focusing on a single piece of information alone can often lead to an incorrect diagnosis of a problem.

What I attempt to offer in this article is not a set of stone tablets, prescribing the exact steps to take to resolve all SQL Server problems, but rather a basic approach and set of tools that have served me well time and again in the six years I've spent working with SQL Server, troubleshooting performance problems. It covers a high-level description of my basic approach, followed by more detailed sections on each of my areas of focus, including wait statistics, virtual file statistics, SQL Server-related performance counters, and plan cache analysis.

Defining a Troubleshooting Methodology

Defining a troubleshooting methodology is hard, because the actual methodology that I apply depends entirely on the specific problem that I am trying to troubleshoot for a specific environment. However, my basic approach, and the tools I use, remain constant, regardless of whether the problem is users complaining of slow performance, or if I am just performing a standard server health check.

When I am examining a server for the first time, I need to establish a picture of its general health, and there are a number of items on which I'll focus in order to obtain this picture. For each piece of information I collect, I'll be examining it in relation to the previous data points, in order to validate or disprove any previous indicators as to the nature of the problem.

Fairly early on in any analysis, I'll take a look at the wait statistics, in the `sys.dm_os_wait_stats` Dynamic Management View (DMV), to identify any major resource waits in the system, at the operating system level. Let's say I identify very high `PAGEIOLATCH_SH` waits, which indicates that sessions are experiencing delays in obtaining a latch for a buffer page. This happens when lots of sessions, or maybe one session in particular, are requesting a lot of data pages that are not available in the buffer pool (and so physical I/O is needed to retrieve them). SQL Server must allocate a buffer page for each one, and place a latch on that page while it's retrieved from disk. The bottleneck here is disk I/O; the disk subsystem simply can't return pages quickly enough to satisfy all of the page requests, and so sessions are waiting for latches, and performance is suffering. However, this does not necessarily mean that a slow disk subsystem is the cause of the bottleneck; it may simply be the victim of excessive I/O caused by a problem elsewhere in the system.

At this point, I'll want to validate this information by examining the virtual file stats, in `sys.dm_io_virtual_file_stats`. Specifically, I'll be looking for evidence of high latency associated with the read and write operations being performed by SQL Server. At the same time, I'll be drilling deeper into the problem, since the virtual file stats will tell me how much I/O activity is being performed by SQL Server, and how the I/O load is distributed across files and databases on the SQL Server instance. To corroborate this data further, I may also check the values of the `Physical Disk\Avg. Disk Reads/sec` and `Physical Disk\Avg. Disk Writes/sec` PerfMon counters. So, at this stage, let's say I've confirmed high latency associated with read and write operations, and found that a particular database is experiencing a very high level of mainly read-based I/O.

My next step will be to investigate the execution statistics for queries against this database, which are maintained in `sys.dm_exec_query_stats` for the execution plans that are in the plan cache. I'll identify the queries that have the highest accumulated physical reads, and then review their associated execution plans, looking for any performance tuning opportunities, either by adding missing indexes to the database, or making changes to the SQL code, in order to optimize the way the database engine accesses the data.

It may be that the code is optimized as far as it can be, but a commonly executed reporting query simply needs to read 6 GB of data from the database, for aggregation, as a part of its execution. If most of this data isn't found in the buffer cache, it will cause high physical I/O, and will account for the high `PAGEIOLATCH_SH` waits. At this point, we may need to look at our hardware configuration and see if the actual root of our problem is a lack of memory installed in the server. In order to verify this, I'll examine the PerfMon memory counters. If I see that the `Page Life`

`Expectancy` is consistently fluctuating, and the system is experiencing non-zero values for `Free List Stalls/sec`, and high `Lazy Writes/sec`, then I can be fairly certain that the buffer pool for the instance is inadequately sized for the amount of data that is being used by the workload. This does not necessarily mean the server needs more memory; it may be that the queries are inefficient and are reading far more data than necessary. To identify the appropriate fix will require further and deeper analysis. This is just one of many possible examples, but it is a real-world example that I have encountered on many occasions while troubleshooting performance problems with SQL Server.

There are a number of points in this troubleshooting process where it would have been very easy to jump to the wrong conclusion regarding the nature of the problem. For example, after reviewing the virtual file statistics and the performance counters for the Physical Disks in the server, it would be easy to conclude that the disk I/O subsystem for the server was inappropriately sized for the amount of work being done, and that additional disks needed to be purchased to handle the disk I/O demands for the server. Unfortunately, scaling up a disk I/O subsystem can be an extremely expensive solution if the problem happens to be a missing index related to a commonly executed query, or buffer pool memory pressure. It is possible that buying a large enough disk configuration will temporarily mask the problem, but since the underlying root cause has not been resolved, you can be sure that the same problem will recur later, as the system continues to grow.

Having provided an overview of my basic approach, the following sections will drill a little deeper into the specific areas of focus, such as wait statistics, virtual file statistics, performance counters, and plan cache usage. I'll explain the information they offer individually, and how all of this information interrelates, to help you assemble a complete understanding of what is going on inside of a server.

> **Don't forget the obvious:**
>
> Just a gentle reminder: before you get yourself all giddy collecting diagnostic data, make sure you've checked for obvious problems. If a user reports that their application is "not working properly," the first thing you should probably do is to ensure that the SQL Server services are actually running on your server. If you open **SQL Server Configuration Manager (SSCM)** and find that the status of the **Database Engine** service, which has a **Service Type** of SQL Server, is **Stopped** then this is very likely the cause of the problem, unless the instance is running in a failover cluster, at which point you need to look at the **Failover Cluster Manager** to identify if the service and its dependent resources are online, and begin troubleshooting why the service fails to start, based on what you find!

Wait Statistics: the Basis for Troubleshooting

One of the first items that I check, when troubleshooting performance problems on a SQL Server, is the wait statistics, which are tracked by the SQLOS during normal operations of any SQL Server.

The SQLOS is a pseudo-operating system that runs as a part of the SQL Server database engine and provides thread scheduling, memory management, and other functions to SQL Server. Normally, such services would, for any processes running inside the operating system, be provided by the operating system. The reason that SQL Server provides its own pseudo-operating system environment is that SQL Server knows how to schedule its tasks better than the Windows operating system does, and the cooperative scheduling that is implemented by the SQLOS allows for higher levels of concurrency than the preemptive scheduling provided by the Windows operating system.

As an example of this, any time that SQL Server has to wait while executing an operation or statement, the time spent waiting is tracked by the SQLOS, as **wait time**. This data is exposed, for each instance of SQL Server installed on a server, in the `sys.dm_os_wait_stats` DMV. The cause and length of the various waits that SQL Server is experiencing can provide significant insight into the cause of the performance problems, as long as you understand exactly what the wait statistics are telling you, and know how to correlate the wait information with the additional troubleshooting information such as the PerfMon counters, and other DMVs.

One of the reasons that wait statistics is such a good place to begin troubleshooting SQL Server performance problems is that, often times, the specifics of the problem are not well defined by the users, when reporting the problem. More often than not, the description of the problem is limited to, "x, y, or z process is slower than normal, can you fix it?" One of the easiest ways to troubleshoot an unknown problem with performance is to look at where and why SQL Server actually had to wait to continue execution of its various tasks.

Usually, Windows Server and SQL Server patches will have been regularly applied to the server, so you'll know how long ago the server was restarted, and therefore over what period the statistics have accumulated (unless someone manually cleared them out – see later). Ideally, you'll want this period to be longer than around two weeks (in order to ensure the stats cover the entire workload), but not so long that the data becomes hard to analyze. In the latter case, you might also consider capturing the values, waiting a period, capturing again and comparing the two.

Diagnosing wait statistics for a single instance of SQL Server is no small task. Often times, the information provided by the wait statistics is only a symptom of the actual problem. To use this wait information effectively, you need to understand the difference between resource (i.e. traceable to a hardware resource) and non-resource waits in the system, and the other outputs provided by SQL Server, in relation to the wait information that is being tracked by the SQL Server instance overall.

As a part of the normal operations of SQL Server, a number of wait conditions exist which are non-problematic in nature and generally expected on the server. These wait conditions can generally be queried from the `sys.dm_os_waiting_tasks` DMV for the system sessions, as shown in Listing 1.1.

```
SELECT DISTINCT
        wt.wait_type
FROM    sys.dm_os_waiting_tasks AS wt
        JOIN sys.dm_exec_sessions AS s ON wt.session_id = s.session_id
WHERE   s.is_user_process = 0
```

Listing 1.1: Discovering system session waits.

When looking at the wait statistics being tracked by SQL Server, it's important that these wait types are eliminated from the analysis, allowing the more problematic waits in the system to be identified. One of the things I do as a part of tracking wait information is to maintain a script that filters out the non-problematic wait types, as shown in Listing 1.2.

```
SELECT TOP 10
        wait_type ,
        max_wait_time_ms wait_time_ms ,
        signal_wait_time_ms ,
        wait_time_ms - signal_wait_time_ms AS resource_wait_time_ms ,
        100.0 * wait_time_ms / SUM(wait_time_ms) OVER ( )
                                AS percent_total_waits ,
        100.0 * signal_wait_time_ms / SUM(signal_wait_time_ms) OVER ( )
                                AS percent_total_signal_waits ,
        100.0 * ( wait_time_ms - signal_wait_time_ms )
        / SUM(wait_time_ms) OVER ( ) AS percent_total_resource_waits
FROM    sys.dm_os_wait_stats
WHERE   wait_time_ms > 0 -- remove zero wait_time
        AND wait_type NOT IN -- filter out additional irrelevant waits
( 'SLEEP_TASK', 'BROKER_TASK_STOP', 'BROKER_TO_FLUSH',
  'SQLTRACE_BUFFER_FLUSH','CLR_AUTO_EVENT', 'CLR_MANUAL_EVENT',
  'LAZYWRITER_SLEEP', 'SLEEP_SYSTEMTASK', 'SLEEP_BPOOL_FLUSH',
  'BROKER_EVENTHANDLER', 'XE_DISPATCHER_WAIT', 'FT_IFTSHC_MUTEX',
  'CHECKPOINT_QUEUE', 'FT_IFTS_SCHEDULER_IDLE_WAIT',
  'BROKER_TRANSMITTER', 'FT_IFTSHC_MUTEX', 'KSOURCE_WAKEUP',
  'LAZYWRITER_SLEEP', 'LOGMGR_QUEUE', 'ONDEMAND_TASK_QUEUE',
  'REQUEST_FOR_DEADLOCK_SEARCH', 'XE_TIMER_EVENT', 'BAD_PAGE_PROCESS',
  'DBMIRROR_EVENTS_QUEUE', 'BROKER_RECEIVE_WAITFOR',
  'PREEMPTIVE_OS_GETPROCADDRESS', 'PREEMPTIVE_OS_AUTHENTICATIONOPS',
  'WAITFOR', 'DISPATCHER_QUEUE_SEMAPHORE', 'XE_DISPATCHER_JOIN',
  'RESOURCE_QUEUE' )
ORDER BY wait_time_ms DESC
```

Listing 1.2: Finding the top ten cumulative wait events.

In general, when examining wait statistics, I focus on the top waits, according to `wait_time_ms`, and look out for high wait times associated with the following specific wait types:

- **`CXPACKET`**
  - Often indicates nothing more than that certain queries are executing with parallelism; `CXPACKET` waits in the server are not an immediate sign of problems, although they may be the symptom of another problem, associated with one of the other high value wait types in the instance.
- **`SOS_SCHEDULER_YIELD`**
  - The tasks executing in the system are yielding the scheduler, having exceeded their quantum, and are having to wait in the runnable queue for other tasks to execute. This may indicate that the server is under CPU pressure.
- **`THREADPOOL`**
  - A task had to wait to have a worker bound to it, in order to execute. This could be a sign of worker thread starvation, requiring an increase in the number of CPUs in the server, to handle a highly concurrent workload, or it can be a sign of blocking, resulting in a large number of parallel tasks consuming the worker threads for long periods.
- **`LCK_*`**
  - These wait types signify that blocking is occurring in the system and that sessions have had to wait to acquire a lock of a specific type, which was being held by another database session. This problem can be investigated further using, for example, the information in the `sys.dm_db_index_operational_stats`.
- **`PAGEIOLATCH_*, IO_COMPLETION, WRITELOG`**
  - These waits are commonly associated with disk I/O bottlenecks, though the root cause of the problem may be, and commonly is, a poorly performing query that is consuming excessive amounts of memory in the server. `PAGEIOLATCH_*` waits are specifically associated with delays in being able to read or write data from the database files. `WRITELOG` waits are related to issues with writing to log files. These waits should be evaluated in conjunction with the virtual file statistics as well as Physical Disk performance counters, to determine if the problem is specific to a single database, file, or disk, or is instance wide.
- **`PAGELATCH_*`**
  - Non-I/O waits for latches on data pages in the buffer pool. A lot of times `PAGELATCH_*` waits are associated with allocation contention issues. One of the best-known allocations issues associated with `PAGELATCH_*` waits occurs in `tempdb` when the a large number of objects are being created and destroyed in `tempdb` and the system experiences contention on the Shared Global Allocation Map (SGAM), Global Allocation Map (GAM), and Page Free Space (PFS) pages in the `tempdb` database.
- **`LATCH_*`**
  - These waits are associated with lightweight short-term synchronization objects that are used to protect access to internal caches,

but not the buffer cache. These waits can indicate a range of problems, depending on the latch type. Determining the specific latch class that has the most accumulated wait time associated with it can be found by querying the `sys.dm_os_latch_stats` DMV.

- **`ASYNC_NETWORK_IO`**
  - This wait is often incorrectly attributed to a network bottleneck. In fact, the most common cause of this wait is a client application that is performing row-by-row processing of the data being streamed from SQL Server as a result set (client accepts one row, processes, accepts next row, and so on). Correcting this wait type generally requires changing the client-side code so that it reads the result set as fast as possible, and then performs processing.

These basic explanations of each of the major wait types won't make you an expert on wait type analysis, but the appearance of any of these wait types high up in the output of Listing 1.2 will certainly help direct your subsequent investigations. For example, if you see `PAGEIOLATCH_*` waits you will probably want to make your next focus the virtual file stats, as explained in the previous example.

Conversely, if the primary wait types in the system are `LCK_*` waits, then you won't want to waste time looking at the disk I/O configuration, but instead focus on discovering what might be causing blocking inside the databases on the server. When `LCK_*` wait types crop up, I tend to jump immediately into more advanced troubleshooting of that specific problem, and begin looking at blocking in the system using the `sys.dm_exec_requests` DMV, and other methods, rather than strictly adhering to my normal methodology. However I may, depending on what I find, double back to see what other problems are in the system.

After fixing any problem in the server, in order to validate that the problem has indeed been fixed, the wait statistics being tracked by the server can be reset using the code in Listing 1.3.

```
DBCC SQLPERF('sys.dm_os_wait_stats', clear)
```

Listing 1.3: Clearing the wait statistics on a server.

One of the caveats associated with clearing the wait statistics on the server, is that it will take a period of time for the wait statistics to accumulate to the point that you know whether or not a specific problem has been addressed.

Virtual File Statistics

A common trap in my experience, when using wait statistics as a primary source of troubleshooting data, is that most SQL Servers will demonstrate signs of what looks like a disk I/O bottleneck. Unfortunately, the wait statistics don't tell you what is causing the I/O to occur, and it's easy to misdiagnose the root cause.

This is why an examination of the virtual file statistics, alongside the wait statistics, is almost always recommended. The virtual file statistics are exposed through the `sys.dm_io_virtual_file_stats` function which, when passed a `file_id` (and possibly `database_id`), will provide cumulative physical I/O statistics, the number of reads and writes on each data file, and the number of reads and writes on each log file, for the various databases in the instance, from which can be calculated the ratio of reads to writes. This also shows the number of I/O stalls and the stall time associated with the requests, which is the total amount of time sessions have waited for I/O to be completed on the file.

```
SELECT  DB_NAME(vfs.database_id) AS database_name ,
        vfs.database_id ,
        vfs.FILE_ID ,
        io_stall_read_ms / NULLIF(num_of_reads, 0) AS avg_read_latency ,
        io_stall_write_ms / NULLIF(num_of_writes, 0)
                                                AS avg_write_latency ,
        io_stall_write_ms / NULLIF(num_of_writes + num_of_writes, 0)
                                                AS avg_total_latency ,
        num_of_bytes_read / NULLIF(num_of_reads, 0)
                                                AS avg_bytes_per_read ,
        num_of_bytes_written / NULLIF(num_of_writes, 0)
                                                AS avg_bytes_per_write ,
        vfs.io_stall ,
        vfs.num_of_reads ,
        vfs.num_of_bytes_read ,
        vfs.io_stall_read_ms ,
        vfs.num_of_writes ,
        vfs.num_of_bytes_written ,
        vfs.io_stall_write_ms ,
        size_on_disk_bytes / 1024 / 1024. AS size_on_disk_mbytes ,
        physical_name
FROM    sys.dm_io_virtual_file_stats(NULL, NULL) AS vfs
        JOIN sys.master_files AS mf ON vfs.database_id = mf.database_id
                                       AND vfs.FILE_ID = mf.FILE_ID
ORDER BY avg_total_latency DESC
```

Listing 1.4: Virtual file statistics.

What I'm primarily looking at in the results is patterns of activity on the file, whether heavy-read or heavy-write, and at the average latency associated with the I/O, as this will direct further investigation and possible solutions.

If the data and log files are on a shared disk array in the server, and the calculated `avg_total_latency` is the same across all of the databases, and higher than what is acceptable for the specific workload, then the problem may be that the workload has outgrown the disk I/O subsystem.

However, if the server hosts a database that is used for archiving data to slower storage, for year-on-year reporting, then it may be that having `PAGEIOLATCH_*` waits in the database is entirely normal, and the `io_stall` information for the specific database files may lead us to determine that the waits are most likely attributable to the archiving process. This highlights the fact that it helps to have a sound knowledge of the underlying configuration and type of workload for the server, while you're troubleshooting the problem.

If a particular file is subject to very heavy read activity (for example a ratio of 10:1, or higher, for the read:write ratio), and is showing high average latency, then I may recommend a RAID change for the disk array, for example from RAID 10 to RAID 5, offering more spindles to share the read I/O.

Hopefully, this discussion has highlighted the key element of effective troubleshooting, which is that you need to examine many "data points" together, in order to arrive at a true diagnosis. The discovery of I/O pressure, revealed by high I/O-related waits, could be caused by inadequate capacity or configuration of the disk subsystem, but its root cause is actually more likely to lie elsewhere, such as in a memory bottleneck in the buffer pool, or excessive index and/or table scans due to poorly written queries (covered in the *Plan Cache Usage* section) and a lack of indexing.

Performance Counters

Many articles, white papers, and blog posts on the Internet attempt to provide detailed lists of the important performance counters that should be monitored for SQL Server instances, along with general guidelines for acceptable values for these counters. However, if you try to collect and analyze the values for all of the available counters, you'll quickly find it an overwhelming task.

Personally, at least in the initial stages of my investigation, I rely on a small subset of counters, directly related to SQL Server. At a more advanced stage in the troubleshooting process, I may also begin collecting Windows counters, in order to verify the information that I already have, or to help isolate an edge case problem to a specific cause.

One of my favorite tools, when I get to the point that I need to collect a larger subset of counters, collecting information from Windows as well as SQL Server, is the **Performance Analysis of Logs** (PAL) tool, which has been made available by Microsoft for free on http://pal.codeplex.com.

The tool provides built-in templates that can be exported to create a **Performance Collector Set** in Windows, each set containing the key counters for a specific product. It includes a template for SQL Server 2005 and 2008. The greatest benefit of this tool is that it also has built-in threshold templates that can be used to process the performance counter data after it has been collected. These can be used to produce a detailed report, breaking down the data into time slices and so automating the analysis of the data into periods of time and activity. If you want to know more about all of the counters related to SQL Server performance, what they mean, and what Microsoft currently says the threshold values for those counters are, I would recommend downloading the tool and taking a look at all the information contained in the SQL Server threshold file.

Nevertheless, the counters I investigate initially are limited to those related to specific areas of SQL Server, and are ones that have proven themselves over the years to provide information critical to determining how to continue with the troubleshooting process. The counters are all available from within SQL Server through the `sys.dm_os_performance_counters` DMV and can be queried using T-SQL alone.

One of the challenges with querying the raw performance counter data directly is that some of the performance counters are cumulative ones, increasing in value as time progresses, and analysis of the data requires capturing two snapshots of the data and then calculating the difference between the snapshots. The query in Listing 1.5 performs the snapshots and calculations automatically, allowing the output to be analyzed directly. There are other performance counters, not considered in Listing 1.5, which have a secondary, associated base counter by which the main counter has to be divided to arrive at its actual value.

```
-- Capture the first counter set
SELECT  CAST(1 AS INT) AS collection_instance ,
        OBJECT_NAME ,
        counter_name ,
        instance_name ,
        cntr_value ,
        cntr_type ,
        CURRENT_TIMESTAMP AS collection_time
INTO    #perf_counters_init
FROM    sys.dm_os_performance_counters
WHERE   ( OBJECT_NAME = 'SQLServer:Access Methods'
          AND counter_name = 'Full Scans/sec'
        )
        OR ( OBJECT_NAME = 'SQLServer:Access Methods'
             AND counter_name = 'Index Searches/sec'
           )
        OR ( OBJECT_NAME = 'SQLServer:Buffer Manager'
             AND counter_name = 'Lazy Writes/sec'
           )
        OR ( OBJECT_NAME = 'SQLServer:Buffer Manager'
```

```
            AND counter_name = 'Page life expectancy'
          )
        OR ( OBJECT_NAME = 'SQLServer:General Statistics'
             AND counter_name = 'Processes Blocked'
          )
        OR ( OBJECT_NAME = 'SQLServer:General Statistics'
             AND counter_name = 'User Connections'
          )
        OR ( OBJECT_NAME = 'SQLServer:Locks'
             AND counter_name = 'Lock Waits/sec'
          )
        OR ( OBJECT_NAME = 'SQLServer:Locks'
             AND counter_name = 'Lock Wait Time (ms)'
          )
        OR ( OBJECT_NAME = 'SQLServer:SQL Statistics'
             AND counter_name = 'SQL Re-Compilations/sec'
          )
        OR ( OBJECT_NAME = 'SQLServer:Memory Manager'
             AND counter_name = 'Memory Grants Pending'
          )
        OR ( OBJECT_NAME = 'SQLServer:SQL Statistics'
             AND counter_name = 'Batch Requests/sec'
          )
        OR ( OBJECT_NAME = 'SQLServer:SQL Statistics'
             AND counter_name = 'SQL Compilations/sec'
          )

-- Wait on Second between data collection
WAITFOR DELAY '00:00:01'

-- Capture the second counter set
SELECT  CAST(2 AS INT) AS collection_instance ,
        OBJECT_NAME ,
        counter_name ,
        instance_name ,
        cntr_value ,
        cntr_type ,
        CURRENT_TIMESTAMP AS collection_time
INTO    #perf_counters_second
FROM    sys.dm_os_performance_counters
WHERE   ( OBJECT_NAME = 'SQLServer:Access Methods'
          AND counter_name = 'Full Scans/sec'
        )
        OR ( OBJECT_NAME = 'SQLServer:Access Methods'
             AND counter_name = 'Index Searches/sec'
          )
        OR ( OBJECT_NAME = 'SQLServer:Buffer Manager'
             AND counter_name = 'Lazy Writes/sec'
          )
        OR ( OBJECT_NAME = 'SQLServer:Buffer Manager'
             AND counter_name = 'Page life expectancy'
          )
        OR ( OBJECT_NAME = 'SQLServer:General Statistics'
             AND counter_name = 'Processes Blocked'
          )
        OR ( OBJECT_NAME = 'SQLServer:General Statistics'
             AND counter_name = 'User Connections'
          )
        OR ( OBJECT_NAME = 'SQLServer:Locks'
             AND counter_name = 'Lock Waits/sec'
          )
        OR ( OBJECT_NAME = 'SQLServer:Locks'
             AND counter_name = 'Lock Wait Time (ms)'
          )
        OR ( OBJECT_NAME = 'SQLServer:SQL Statistics'
             AND counter_name = 'SQL Re-Compilations/sec'
          )
        OR ( OBJECT_NAME = 'SQLServer:Memory Manager'
             AND counter_name = 'Memory Grants Pending'
```

```
            )
        OR ( OBJECT_NAME = 'SQLServer:SQL Statistics'
             AND counter_name = 'Batch Requests/sec'
            )
        OR ( OBJECT_NAME = 'SQLServer:SQL Statistics'
             AND counter_name = 'SQL Compilations/sec'
            )

-- Calculate the cumulative counter values
SELECT  i.OBJECT_NAME ,
        i.counter_name ,
        i.instance_name ,
        CASE WHEN i.cntr_type = 272696576
                  THEN s.cntr_value - i.cntr_value
             WHEN i.cntr_type = 65792 THEN s.cntr_value
        END AS cntr_value
FROM    #perf_counters_init AS i
        JOIN #perf_counters_second AS s
                ON i.collection_instance + 1 = s.collection_instance
                  AND i.OBJECT_NAME = s.OBJECT_NAME
                  AND i.counter_name = s.counter_name
                  AND i.instance_name = s.instance_name
ORDER BY OBJECT_NAME

-- Cleanup tables
DROP TABLE #perf_counters_init
DROP TABLE #perf_counters_second
```

Listing 1.5: SQL Server performance counters.

The performance counters collected by this script are:

- **SQLServer:Access Methods\Full Scans/sec**
- **SQLServer:Access Methods\Index Searches/sec**
- **SQLServer:Buffer Manager\Lazy Writes/sec**
- **SQLServer:Buffer Manager\Page life expectancy**
- **SQLServer:Buffer Manager\Free list stalls/sec**
- **SQLServer:General Statistics\Processes Blocked**
- **SQLServer:General Statistics\User Connections**
- **SQLServer:Locks\Lock Waits/sec**
- **SQLServer:Locks\Lock Wait Time (ms)**
- **SQLServer:Memory Manager\Memory Grants Pending**
- **SQLServer:SQL Statistics\Batch Requests/sec**
- **SQLServer:SQL Statistics\SQL Compilations/sec**
- **SQLServer:SQL Statistics\SQL Re-Compilations/sec**

The two **`Access Methods`** counters provide information about the ways that tables are being accessed in the database. The most important one is the `Full Scans/sec` counter, which can give us an idea of the number of index and table scans that are occurring in the system.

If the disk I/O subsystem is the bottleneck (which, remember, is most often caused by pressure placed on it by a problem elsewhere) and this counter is showing that there are scans occurring, it may be a sign that there are missing indexes, or inefficient code in the database. How many scans are problematic? It depends entirely on the size of the objects being scanned and the type of workload being run. In general, I want the number of `Index Searches/sec` to be higher than the number of `Full Scans/sec` by a factor of 800–1000. If the number of `Full Scans/sec` is too high, I use the Database Engine Tuning Advisor (DTA) or the Missing Indexes feature to determine if there are missing indexes in the database, resulting in excess I/O operations.

The **`Buffer Manager`** and **`Memory Manager`** counters can be used, as a group, to identify if SQL Server is experiencing memory pressure. The values of the `Page Life Expectancy`, `Free List Stalls/sec`, and `Lazy Writes/sec` counters, when correlated, will validate or disprove the theory that the buffer cache is under memory pressure.

A lot of online references will tell you that if the `Page Life Expectancy` (`PLE`) performance counter drops lower than 300, which is the number of seconds a page will remain in the data cache, then you have memory pressure. However, this guideline value for the `PLE` counter was set at a time when most SQL Servers only had 4 GB of RAM, and the data cache portion of the buffer pool was generally 1.6 GB. In modern servers, where it is common for SQL Servers to have 32 GB or more of installed RAM, and a significantly larger data cache, having 1.6 GB of data churn through that cache every 5 minutes is not necessarily a significant event.

In short, the appropriate value for this counter depends on the size of the SQL Server data cache, and a fixed value of 300 no longer applies.

Instead, I evaluate the value for the PLE counter based on the installed memory in the server. To do this, I take the base counter value of 300 presented by most resources, and then determine a multiple of this value based on the configured buffer cache size, which is the 'max server memory' sp_configure option in SQL Server, divided by 4 GB. So, for a server with 32 GB allocated to the buffer pool, the PLE value should be at least (32/4)*300 = 2400.

If the PLE is consistently below this value, and the server is experiencing high Lazy Writes/sec, which are page flushes from the buffer cache outside of the normal CHECKPOINT process, then the server is most likely experiencing data cache memory pressure, which will also increase the disk I/O being performed by the SQL Server. At this point, the Access Methods counters should be investigated to determine if excessive table or index scans are being performed on the SQL Server.

The General Statistics\Processes Blocked, Locks\Lock Waits/sec, and Locks\Lock Wait Time (ms) counters provide information about blocking in the SQL Server instance, at the time of the data collection. If these counters return a value other than zero, over repeated collections of the data, then blocking is actively occurring in one of the databases and we can use tools such as the Blocked Process Report in SQL Trace, or the sys.dm_exec_requests, sys.dm_exec_sessions and sys.dm_os_waiting_tasks DMVs to troubleshoot the problems further.

The three SQL Statistics counters provide information about how frequently SQL Server is compiling or recompiling an execution plan, in relation to the number of batches being executed against the server. The higher the number of SQL Compilations/sec in relation to the Batch Requests/sec, the more likely the SQL Server is experiencing an ad hoc workload that is not making optimal using of plan caching. The higher the number of SQL Re-Compilations/sec in relation to the Batch Requests/sec, the more likely it is that there is an inefficiency in the code design that is forcing a recompile of the code being executed in the SQL Server. In either case, investigation of the Plan Cache, as detailed in the next section, should identify why the server has to consistently compile execution plans for the workload.

The Memory Manager\Memory Grants Pending performance counter provides information about the number of processes waiting on a workspace memory grant in the instance. If this counter has a high value, SQL Server may benefit from additional memory, but there may be query inefficiencies in the instance that are causing excessive memory grant requirements, for example, large sorts or hashes that can be resolved by tuning the indexing or queries being executed.

Plan Cache Usage

In my experience, the Plan Cache in SQL Server 2005 and 2008 is one of the most underused assets for troubleshooting performance problems in SQL Server. As a part of the normal execution of batches and queries, SQL Server tracks the accumulated execution information for each of the plans that is stored inside of the plan cache, up to the point where the plan is flushed from the cache as a result of DDL operations, memory pressure, or general cache maintenance. The execution information stored inside of the plan cache can be found in the sys.dm_exec_query_stats DMV as shown in the example query in Listing 1.6. This query will list the top ten statements based on the average number of physical reads that the statements performed as a part of their execution.

```
SELECT TOP 10
        execution_count ,
        statement_start_offset AS stmt_start_offset ,
        sql_handle ,
        plan_handle ,
        total_logical_reads / execution_count AS avg_logical_reads ,
        total_logical_writes / execution_count AS avg_logical_writes ,
        total_physical_reads / execution_count AS avg_physical_reads ,
        t.TEXT
FROM    sys.dm_exec_query_stats AS s
        CROSS APPLY sys.dm_exec_sql_text(s.sql_handle) AS t
ORDER BY avg_physical_reads DESC
```

Listing 1.6: SQL Server execution statistics.

The information stored in the plan cache can be used to identify the most expensive queries based on physical I/O operations for reads and for writes, or based on different criteria, depending on the most problematic type of I/O for the instance, discovered as a result of previous analysis of the wait statistics and virtual file statistics.

Additionally, the sys.dm_exec_query_plan() function can be cross-applied using the plan_handle column from the sys.dm_exec_query_stats DMV to get the execution plan that is stored in the plan cache. By analyzing these plans, we can identify problematic operations that are candidates for performance tuning.

> **Query performance tuning:**
>
> A full discussion of query performance tuning is beyond the scope of this book. In fact, several notable books have been written on this topic alone, including *"SQL Server 2008 Query Performance Tuning Distilled"* and (*"Inside Microsoft SQL Server 2008: T-SQL Querying"*.

The information in the sys.dm_exec_query_stats DMV can also be used to identify the statements that have taken the most CPU time, the longest execution time, or that have been executed the most frequently.

In SQL Server 2008, two additional columns, `query_hash` and `query_plan_hash`, were added to the `sys.dm_exec_query_stats` DMV. The `query_hash` is a hash over the statement text to allow similar statements to be aggregated together. The `query_plan_hash` is a hash of the query plan shape that allows queries with similar execution plans to be aggregated together. Together, they allow the information contained in this DMV to be aggregated for ad hoc workloads, in order to determine the total impact of similar statements that have different compiled literal values.

Summary

This article has outlined my basic approach to investigating performance problems in SQL Server. This approach is more or less the same, regardless of whether it is a server I know well, or one I'm investigating for the first time, with no prior knowledge of the health and configuration of the SQL Server instance it houses. Based on the information gathered using this methodology, more advanced diagnosis of the identified problem areas can be performed.

The most important point that I want to stress is that no single piece of information in SQL Server should be used to pinpoint any specific problem. The art of taming an unruly SQL Server is the art of assembling the various pieces of the puzzle so that you have a complete understanding of what is going on inside of a server. If you focus only on what is immediately in front of you, you will, in most cases, miss the most important item, which is the true root cause of a particular problem in SQL Server.

This article is adapted, with kind permission of the authors, from the book "Troubleshooting SQL Server: A guide for the Accidental DBA" the publication of which is imminent. It describes in depth the tools and practical techniques for troubleshooting many of the most common causes of SQL Server problems, including high CPU usage, memory mismanagement, missing indexes, blocking, deadlocking, full transactions logs and accidentally-lost data.

SQL Server MVP Deep Dives Vol. 2

Published Thursday, September 29, 2011 10:36 PM

The confetti and silly hats, which are quintessential standards of the celebration of a new year, had just been freshly packed away when an email popped up in my inbox. It was Kalen Delaney announcing the call for submissions for upcoming the ***SQL Server Deep Dives Volume 2*** book. Potential authors were challenged to write about their area of *passion* in regard to SQL Server. I jumped at the chance to offer my contribution. To my delight, my chapter made the cut!

The topic of passion that I contributed was on the topic of personally identifiable data and the super powers that the DBA holds in its protection. I titled it ***Will the real Mr. Smith please stand up?***, in honor of the tag line of old ***To Tell The Truth*** game show. It can be found as the eleventh chapter of the book within the ***Database Administration*** section. It was quite an honor to participate in the authoring of this book and to share these pages with such an esteemed list of SQL Server Gurus.

All royalties from ***SQL Server Deep Dives Volume 2*** will be donated to Operation Smile which is an international children's medial charity. So when you purchase this book, you not only help yourself you also will help a child who suffers from facial deformities start a new life.

For those who will be attending the 2011 PASS Summit on October 11 - 14 in Seattle, Washington, there will be an opportunity to get your hands on a fresh off-the-press copy of this life changing book. There will also be the opportunity to have your copy signed by many of the contributing authors. Others, like myself - who will not make it to the biggest SQL Server event of the year, a copy of this book can be ordered directly from Manning Publishing, Amazon.com and Barnes and Noble shortly after the Summit.

Enjoy!

by Johnm

Team Foundation Server on Azure: First impressions

29 September 2011
by Bahadir Arslan

> Team Foundation Service, the hosted TFS service on Azure, together with Visual Studio 11, has now provided much of the functionality that was missing from the previous incarnation of TFS. Bahadir gives a summary of the new features as seen in the test service, and demonstrates why they are so useful for team-based development

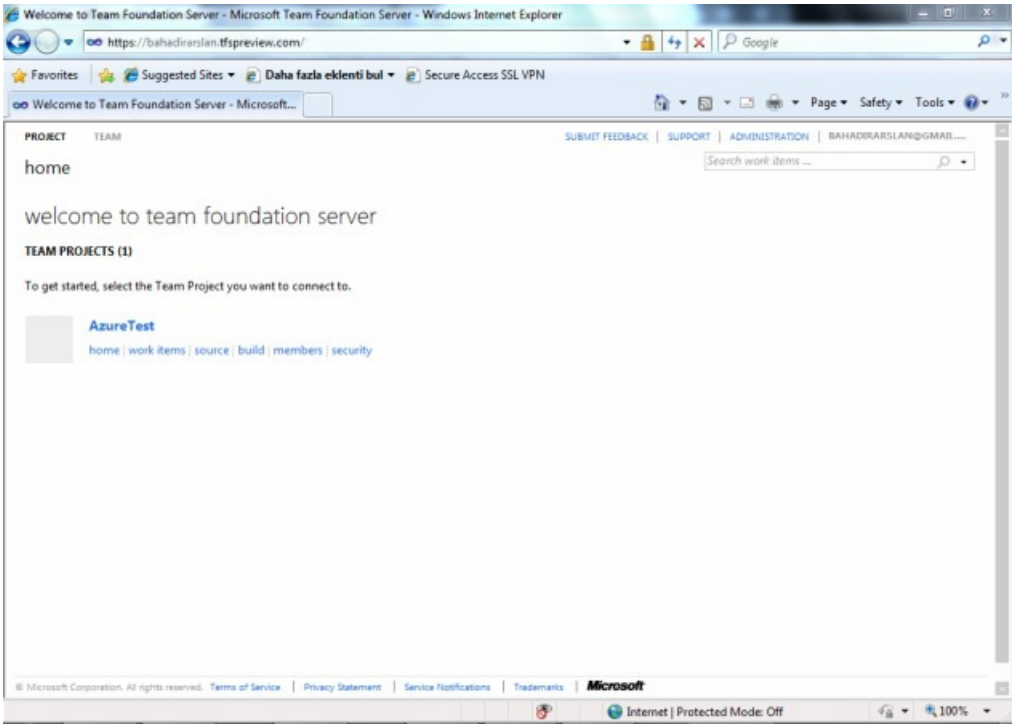
At the 'Build Windows 2011' conference in Anaheim, California, many new Microsoft products were announced in the Keynote presentations, notably Windows 8 and Visual Studio 11. For me though, the introduction of the Team Foundation Service (also known as 'Team Foundation Server on Azure' or 'TFS Service') was the most exciting new product by far.

Finally, TFS on Azure has become a reality. On the second day of the conference an invitation for TFS on Azure was given to all of the attendees. Most of the interested people like me who couldn't be there had to wait an extra two hours while Brian Harry posted a new blog and issued two hundred and fifty more invitations for TFS on Azure. Fortunately I managed to wrangle an invitation which gave me a chance to look more closely at this new product.

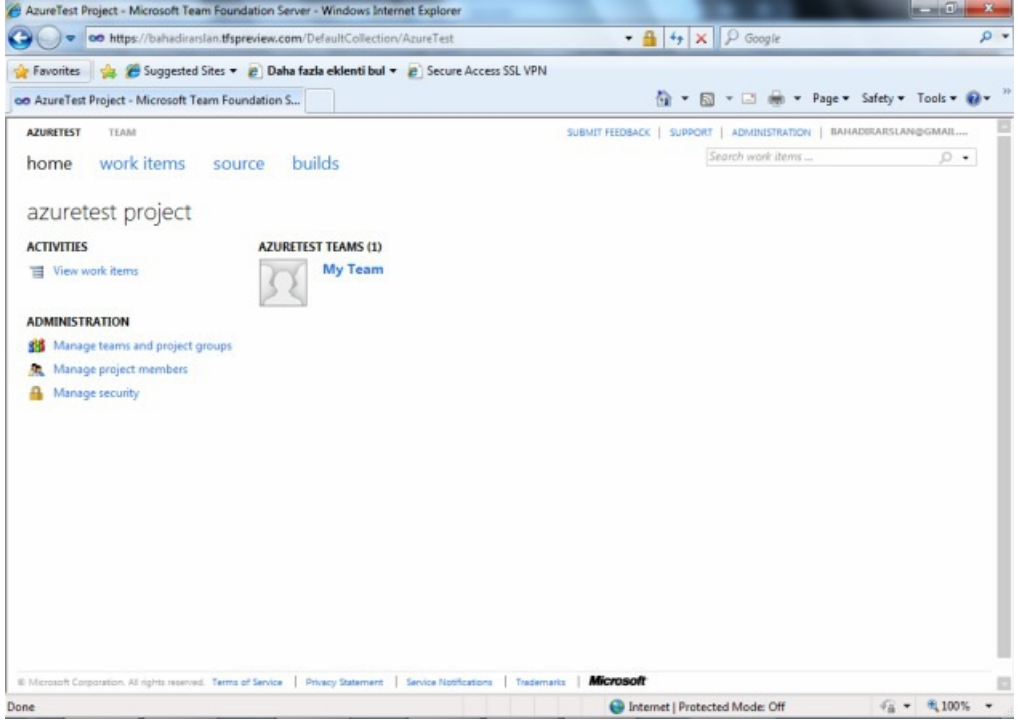
Creating a TFS Service Account and Using the New Web Access

I followed the directions Brian Harry explained in detail in his blog post on how to create an account for this test service and started to use the new Web Access. First, you need to sign in to TFS. As you may know, previously the TFS Service users had to authenticate TFS via their Active Directory accounts; now we are able to authenticate using a Windows Live ID. Additional routes to authentication may become available in the future.

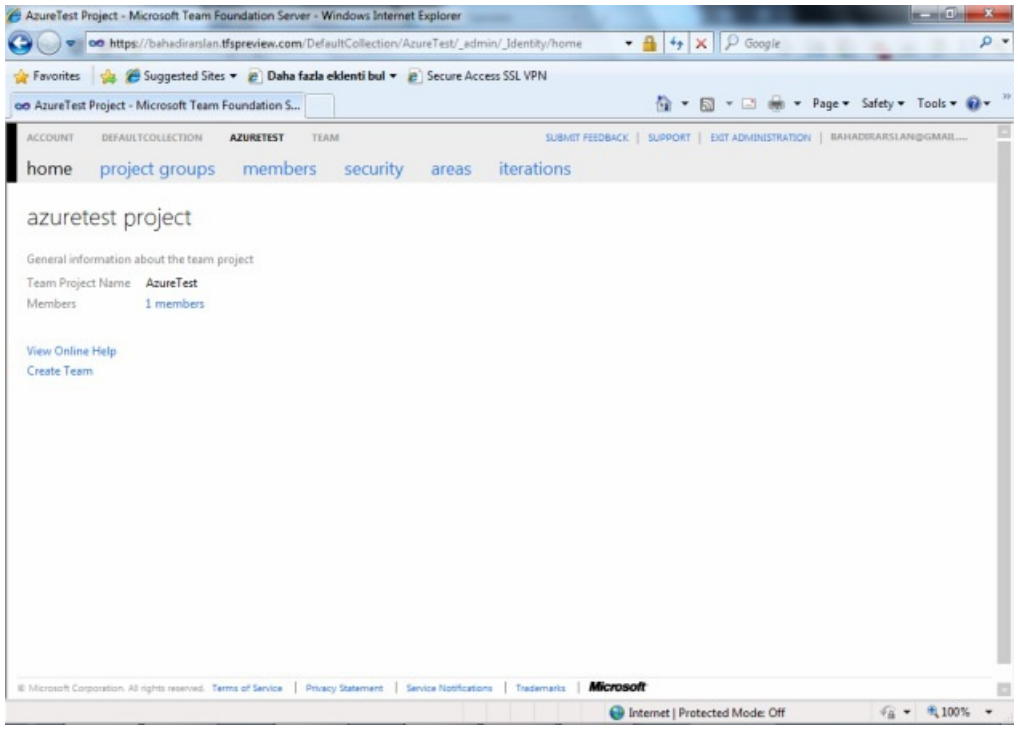
After I signed in, I created a new test project named **AzureTest** with the CMMI 6.0 (Preview) Process Template. Below is the home page of TFS Service Web Access, but as you can see there is only one team project so far.



This new web access is very important for your account because you have control over users, areas, iterations, user rights, source control and work items from this web site. There are two modes of usage, Administration and Normal. When you are in Administration mode, you can configure your team project items such as members, security, areas or iterations, but when you are in Normal mode you use the web site as a normal TFS client. This is an example of Normal mode:



... and Administration mode looks like this:

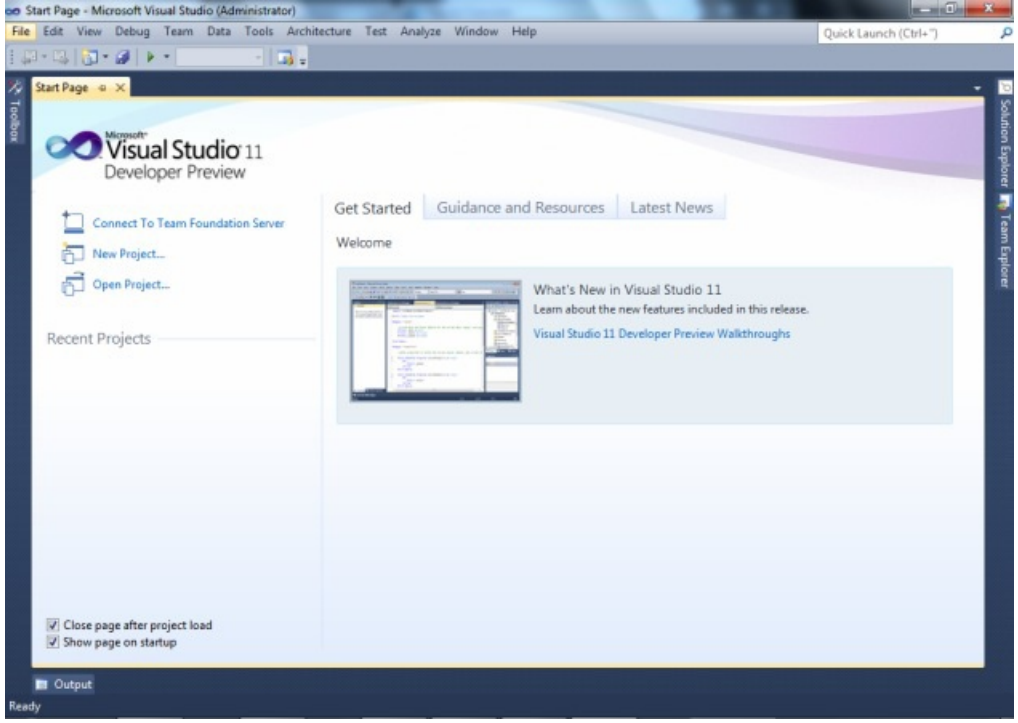


Just to reiterate, you can find more details about Web Access at Brian Harry's blog post.

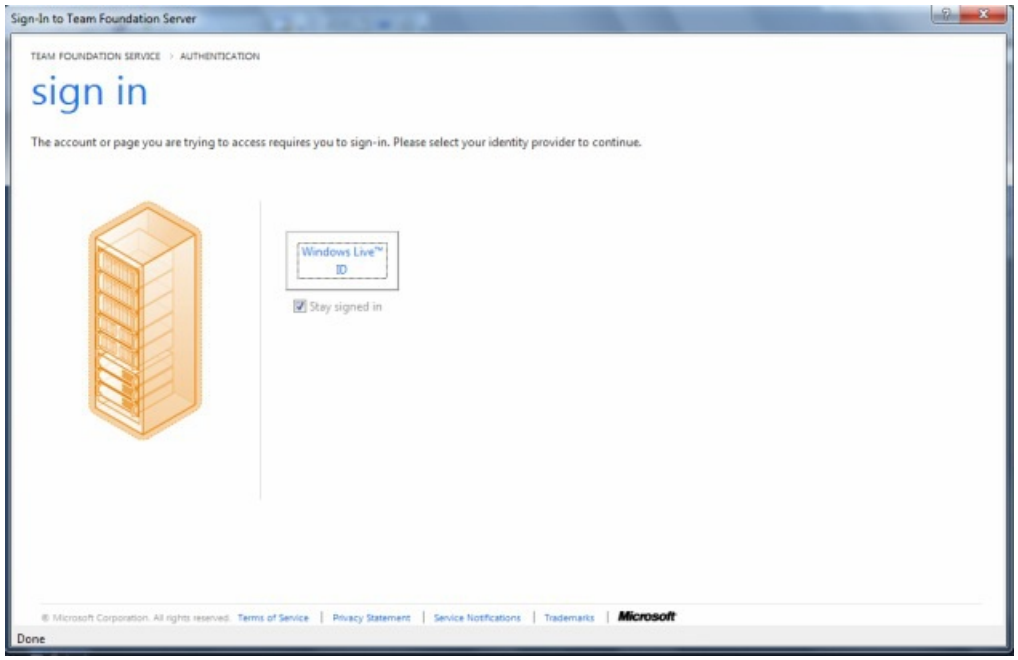
Connecting To TFS Service From Visual Studio

One great advantage here is that you don't need to have Visual Studio 11 to connect to the TFS Service. Microsoft now lets you connect from your current Visual Studio 2010 and any other clients via Visual Studio 2011 Team Explorer Preview. Before you can achieve this though, you'll have to download a client patch. If you want to connect to the TFS Service via Eclipse or any other clients you have to download Team Explorer Everywhere.

In this article I want to focus on the new Visual Studio 11 Developer Preview and TFS Service. Currently Visual Studio 11 Developer Preview can be downloaded from here. After you install and open Visual Studio 11, you'll see the usual Start Page. At this point, click **Connect To Team Foundation Server** at the left side of start page.



After you click **Connect To Team Foundation Server**, the usual **Connect To Team Project** window appears. The first difference you'll notice after you add the TFS Service is the **Sign-In Team Foundation Server** window as can be seen below.

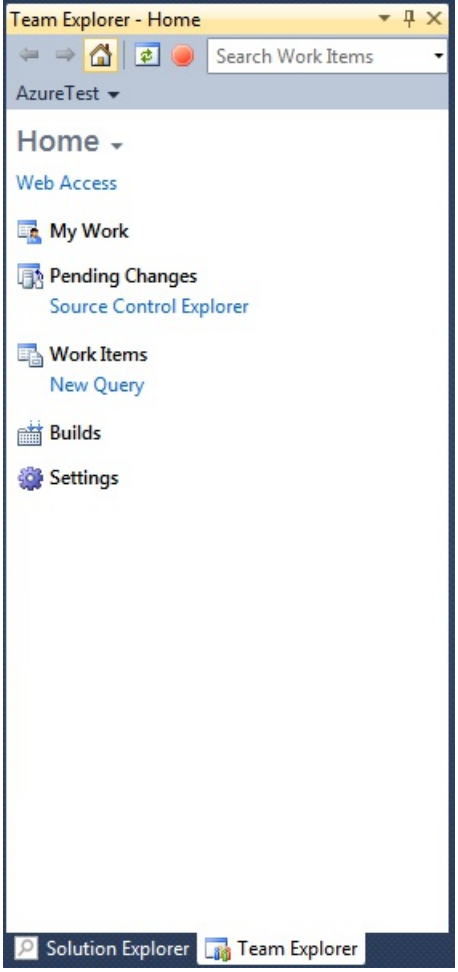


You can sign in with your Windows Live ID; when click the button you'll be redirected to a sign-in page. Once you have successfully signed in, Visual Studio 11 connects to the TFS Service.

New Team Explorer Window

Team Explorer is the most significant change in Visual Studio 11 to accommodate TFS and is your new central point for TFS jobs. In earlier versions of Visual Studio, there was a **Pending Changes** window beside Team Explorer and you had to use both of them to use TFS. But with TFS 11 (and, of course, TFS Service as well), **Pending Changes** has moved into Team Explorer.

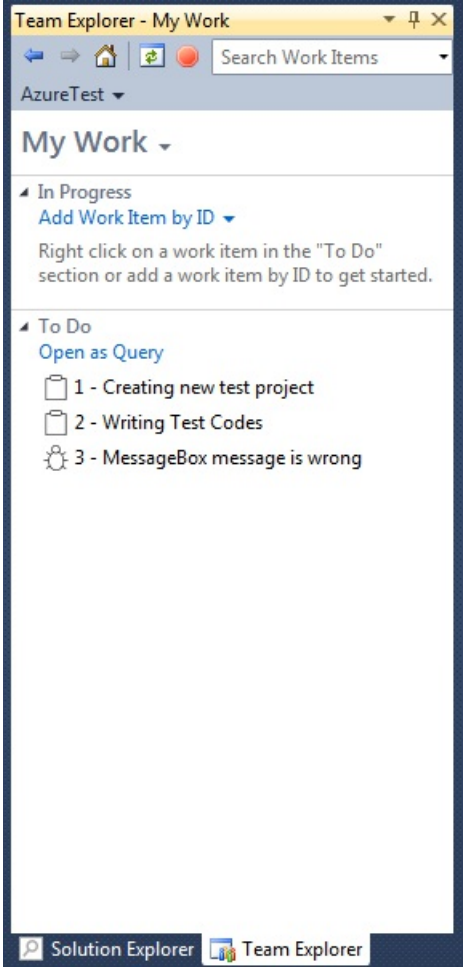
These aren't the only changes to Team Explorer. Since it has been created from the ground up, there is a whole new Team Explorer window which looks like this:



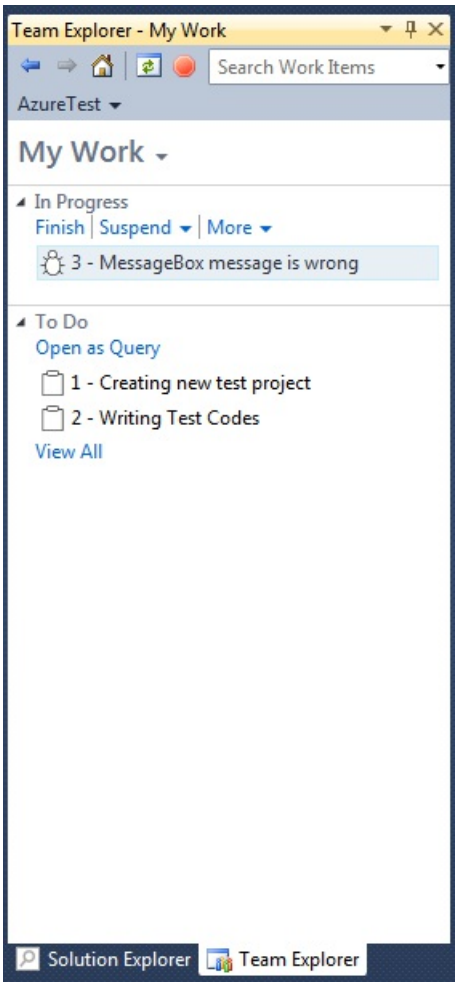
At the top of window, you can see the standard toolbar which lets you navigate between pages. Below this toolbar you can see name of the currently connected team project and a down arrow. Clicking the down arrow opens a menu which lets you select another team project or create a new project.

The page currently open is the Home page, you can access the main views from here. We'll now look more closely at some other views.

My Work



My Work is a new area for Visual Studio users where you can organize your work using **work Items**. For example, in the **To Do** section all of the work items assigned to you are listed. Above the **To Do** section there is an **In Progress** section which contains your active work items. You can add work items here by right clicking on **Work Items** in the **To Do** section or by clicking **Add Work Item by ID** in the **In Progress** section.



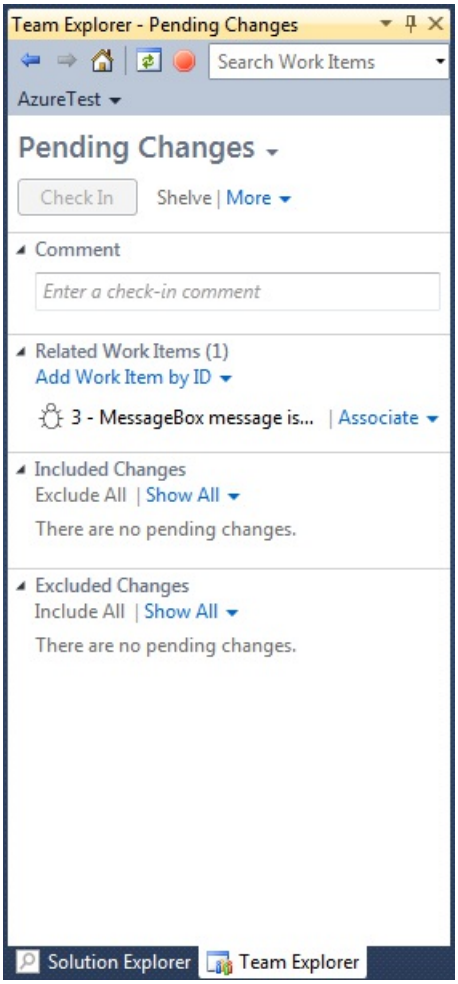
I expected the work item's status to change to **active** when I added it to the **In Progress** section; but it doesn't appear to update automatically. I'm not sure whether or not this is a bug.

There are various other actions that you can perform on work items here. When you click the **Finish** link your pending changes will be checked in, but your work item doesn't appear to automatically update to a new logical workflow state.

You can also perform a **Suspend** action. Sometimes in real life you get surprises which disrupt your workflow. Imagine you are working on a task when a critical bug presents itself. At this point, you need to suspend your current work and move to tackle the critical task. You had two options to deal with this in earlier versions of TFS. The first option was using a fresh workspace; the second option was shelving your current pending changes and starting work on the new task. In TFS 11, you have a new option, suspend your work. This is another way to shelve work, but is much easier.

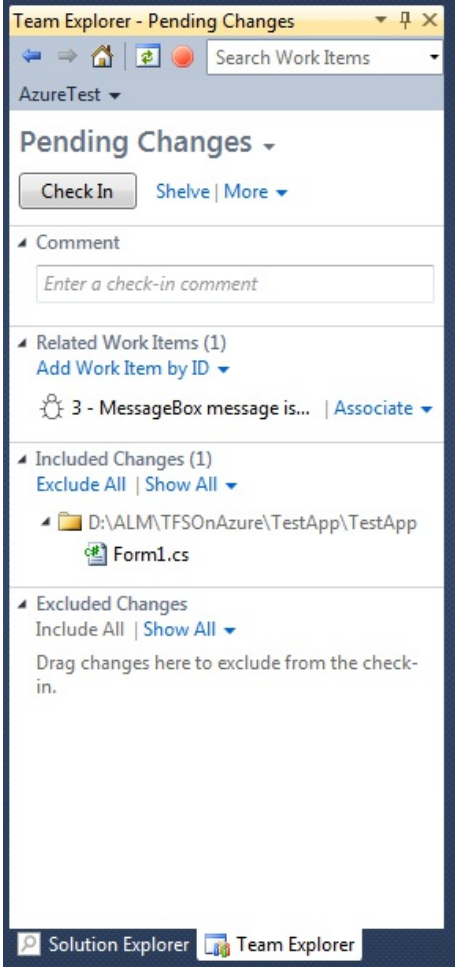
You can also add more work items to the **In Progress** section through the **More** action.

Pending Changes



I've already mentioned that the **Pending Changes** window is integrated into Team Explorer. This new **Pending Changes** view is much easier to use than the old one and is split into four sections. The first one is a **Comment** section as in earlier versions. The second section is called **Related Work Items**. In this section, the work items which are listed will be associated with your changeset when you check-in. By default, those work items that are found in the **In Progress** section of the **My Work** view are listed.

The third section of this view is **Pending Changes**. Currently I have no pending changes so there is nothing in the list. But if I edit some of files, then I can populate this section. Let's try this out:



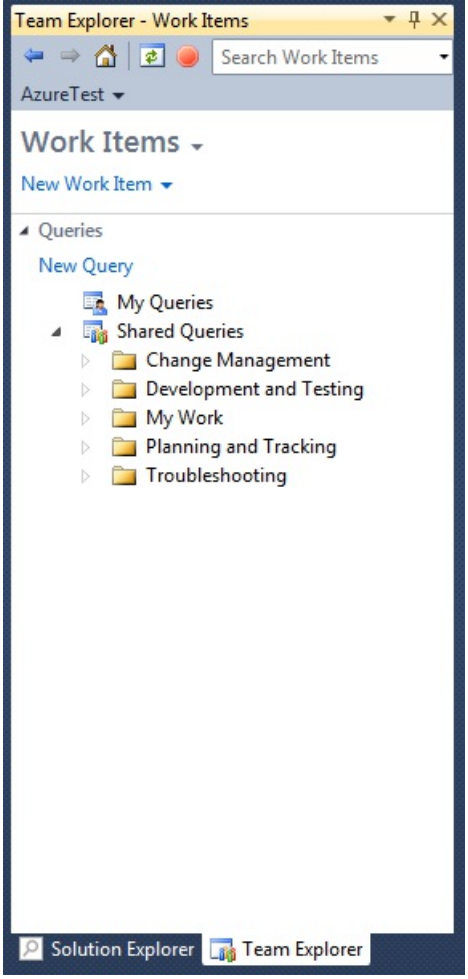
As you can see, this file appears within the **Included Changes** section. Before Visual Studio 11, when you wanted to edit a file that was under source control, you had to check-out that file from Visual Studio first, and then edit it. With Visual Studio 11, you can skip that step. The files under source control are no longer read-only, so you can edit them directly from outside Visual Studio 11. When no longer read-only, so you can edit them directly from outside Visual Studio 11. When you do this, Visual Studio automatically detects what's going on and marks the file as being checked-out. Another big improvement is offline working. When you are working while disconnected from TFS and you modify a file, Visual Studio automatically detects this and marks it as checked-out. Neat.

You might ask at this point, where are the checkboxes? In earlier versions you had to check the checkboxes of the members of the file-list that you want to check-in. But, in this version there is no checkbox; we have an **Excluded Changes** section instead. If you want to exclude a file from check-in, you have to move that file from the **Included Changes** section to the fourth section called **Excluded Changes**.

If we turn back to the top of window we can see the **Check-in** button for checking files into TFS. Near the button there is **Shelve** link, if you click this, the usual **Shelve windows** will appear. Near the **Shelve** link, there is a **More** menu link. Clicking this will open a new menu where you can find other actions such as, **Find Shelvesets**, **Get All Conflicts**, **Undo All**, **Request Review** (this will be detailed later) and finally, **Settings**.

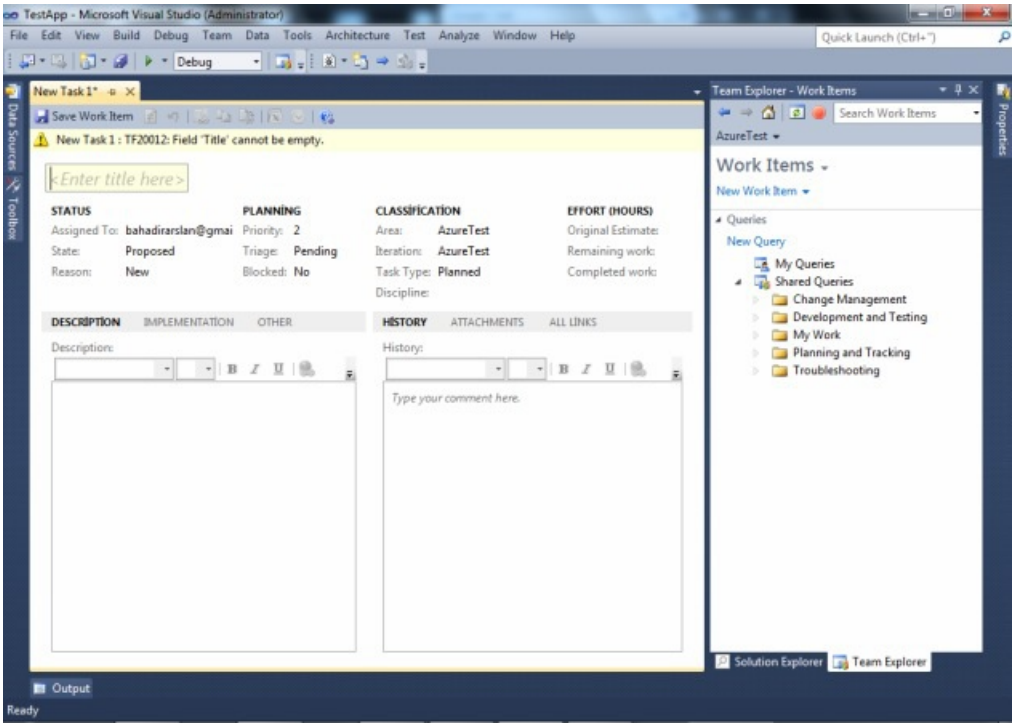
You can find more information about this subject in Brian Harry's blog post.

Work Items

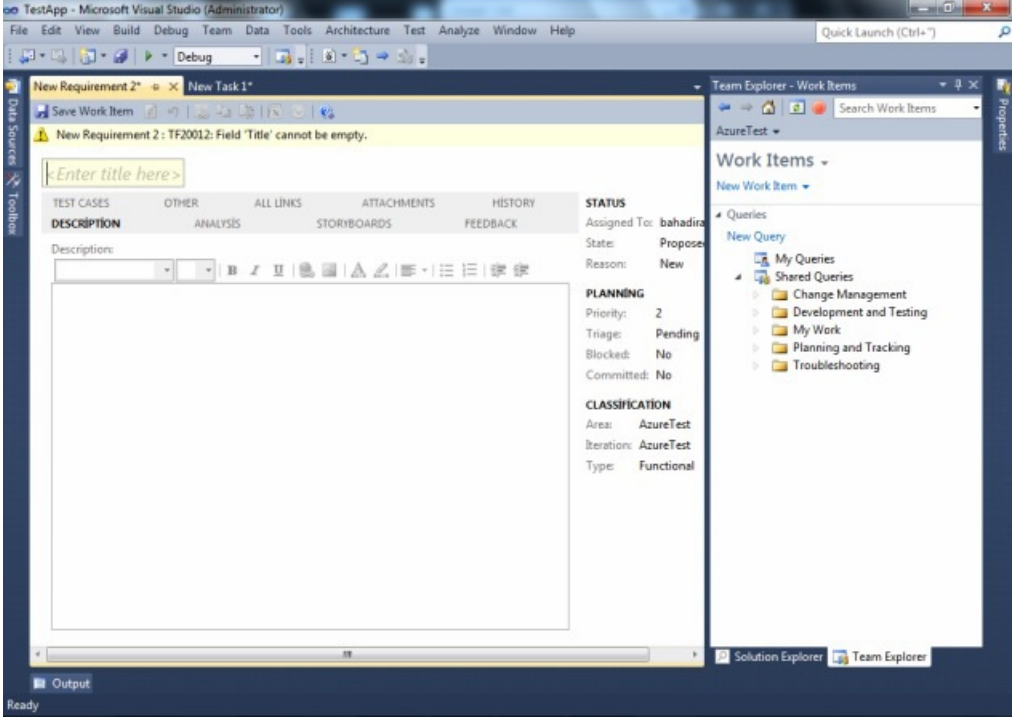


This view looks like the previous version. You can find **New Work Item** link and **Work Items -> Queries** in this view. There are no important changes here, but the work item forms have changed, so let's look them more closely.

New Work Item Forms

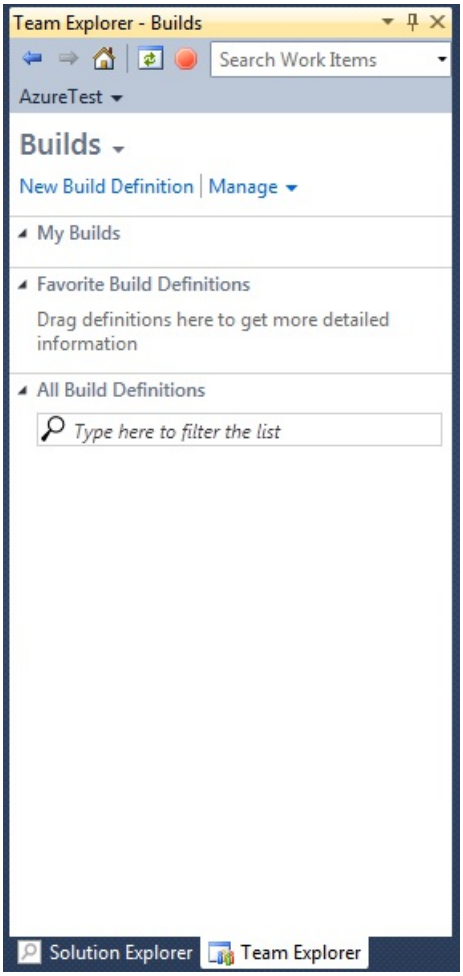


From this image you can see that this is all very different from the previous versions. If you look carefully, some of the words may look a bit funny; this is likely caused by my regional settings being set to Turkish and I reckon it's a non-critical bug. Another thing to notice is we finally get a rich text Description field.



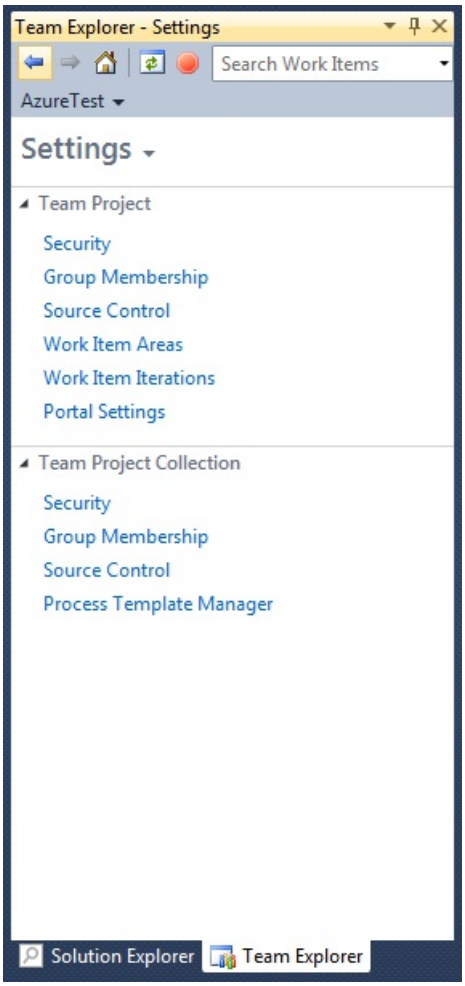
In this screenshot, the Requirement work item is shown and is rather different to the Task type. The design has changed but the individual fields haven't.

Builds



Builds is a new view. Build definitions can be found here, and new build definitions can be defined from here also. In this test TFS Service, there is no default build controller available and I haven't installed a build agent on my computer yet so I couldn't try it out. I am planning to write about this soon with another Team Foundation Server 11 Preview article.

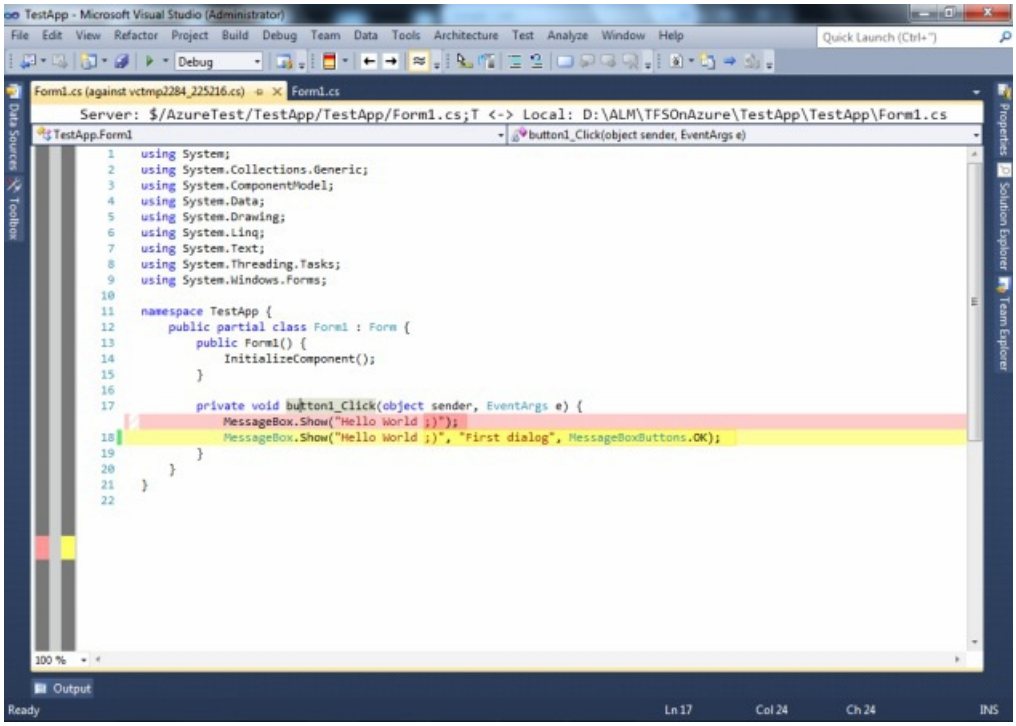
Settings



Settings view contains links for general settings. An array of links is presented to allow web access to operations such as **Security**, **Group Membership**, **Work Item Areas**, and **Work Item Iterations**. Some of these open familiar windows such as **Source Control**, **Portal Settings**, and **Process Template Manager**.

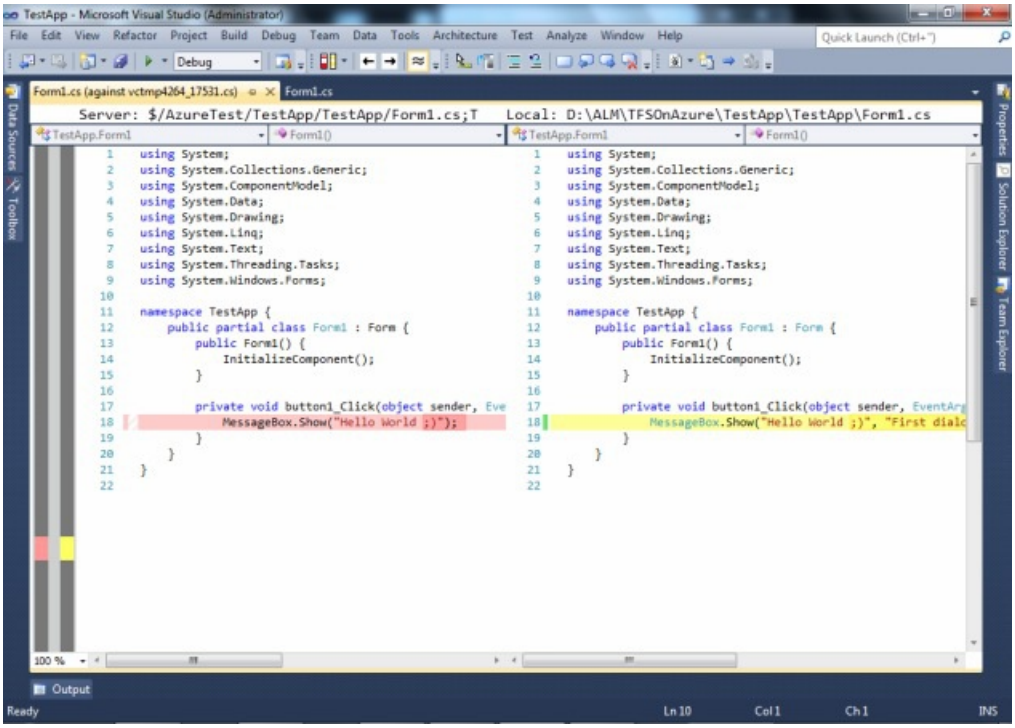
Compare and Merge

In previous versions of TFS, comparing and merging was a big problem. The **Compare** window hadn't changed since VSS, until now. Let's take a closer look:



The **Compare** window has more than one mode to compare, and the default setting is Inline mode. In this mode a red-highlighted line means

source version, a yellow-highlighted line means target version. Main changes are marked as a red-lined square. If you want to see a more familiar mode you can look at the Side By Side mode.



The Side By Side mode probably looks more familiar but actually it's very different from previous versions: Now it is not just gray screen, it' the usual coding view. You can see line numbers, highlights for keywords, or combo boxes for navigating in code. As shown above, there is a little bug at the top of compare window. File paths are in mesh.

There are two more modes, Left file only and Right file only. These modes only show the selected part of compare screen.

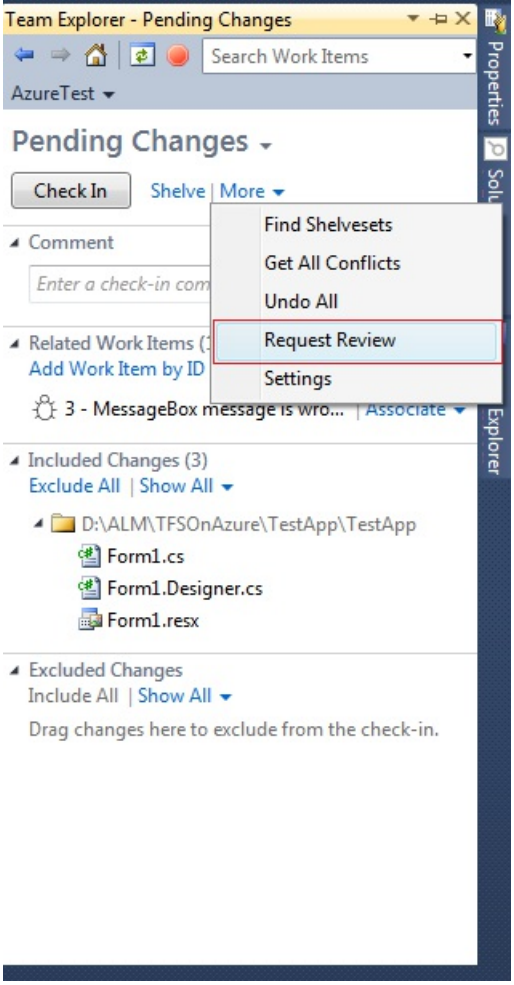
If you look carefully, you can see a new mini toolbar shown when the **Compare** window is active, so let's look this new toolbar more closely:

In this mini toolbar there are four buttons. The first one is the **mode-changing** button to change between the comparison modes I described above. The second and third buttons let you navigate between differences. The last button lets you ignore whitespace while comparing. You can find more information about Merge and Compare at Brian Harry's blog post.

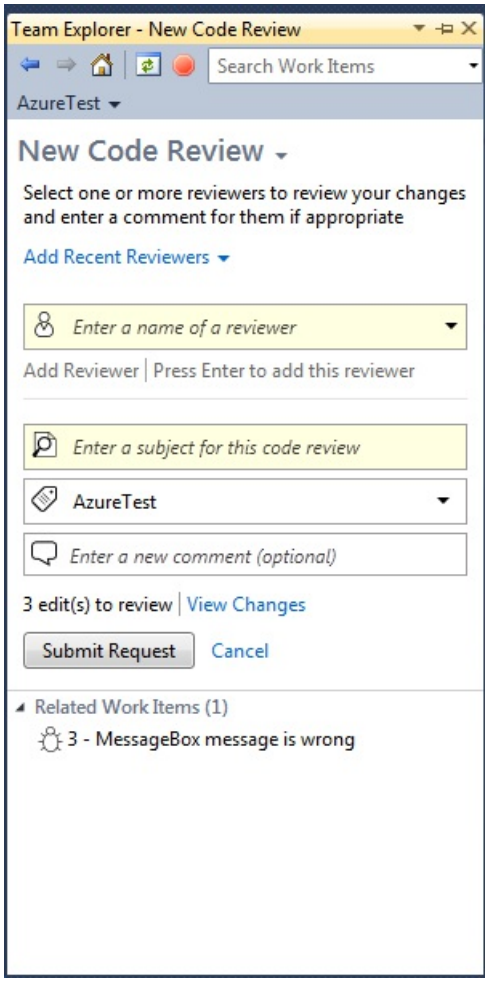
Code Review Workflow

Visual Studio 11 has another new feature called the **Code Review Workflow**. Code reviewing is a very important process for development teams, but there didn't used to be one single tool that suited every requirement. Microsoft has finally developed an integrated solution for this workflow and integrated it into Visual Studio 11. Let's look at this new special workflow.

For this example, I modified a form and went to request a code-review from my colleague. I needed to open the **Pending Changes** view in Team Explorer window and clicked **More -> Request Review**.

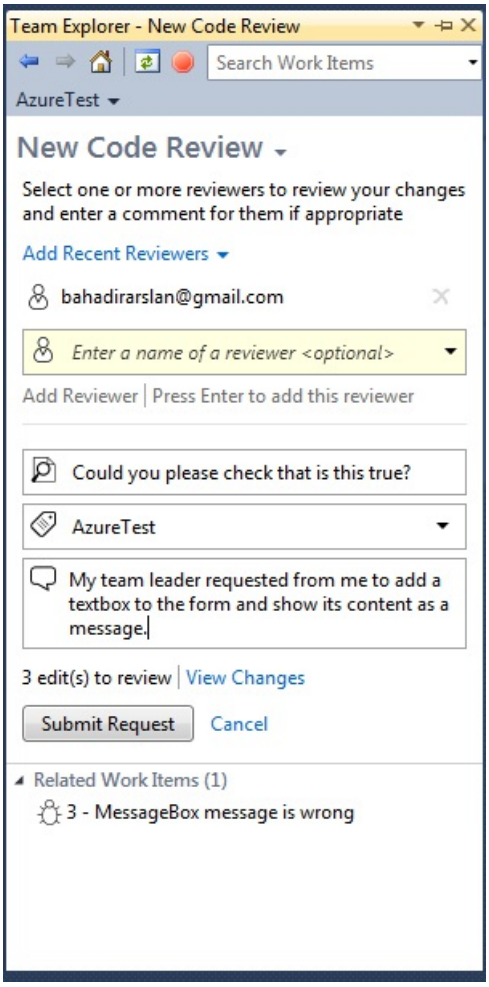


After you click **Request Review**, a new **Code Review** view will be opened.

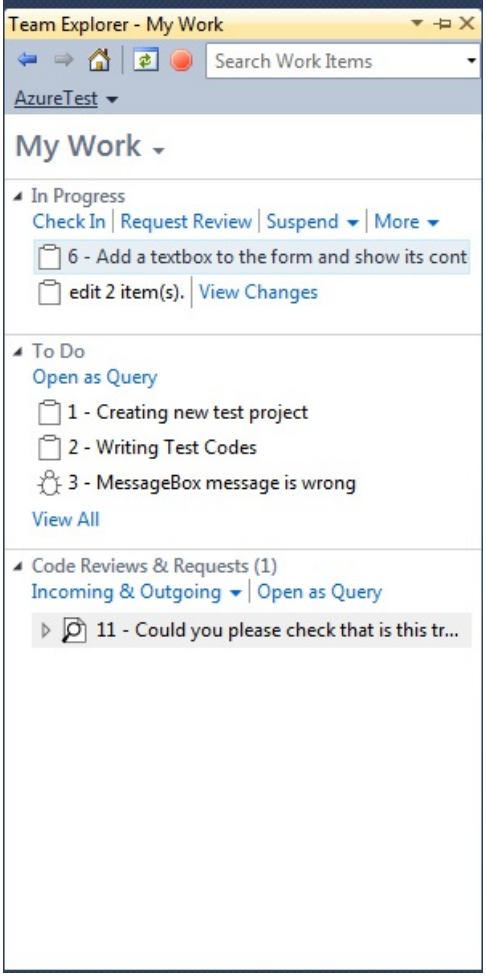


First, you have to enter your colleague's name to do the review. In this example I have to enter my name because there is no user except me. If you

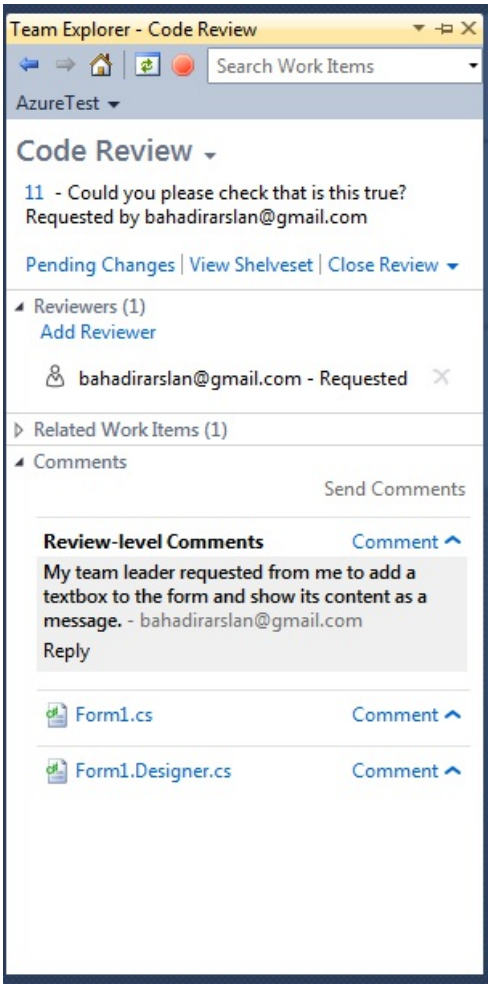
wish, more than one name can be entered. After selecting the reviewer, you then have to enter a subject for this Code Review. The textbox below **Review Name** is for the Area Path. In this project there is only one Area Path so I don't need to select anything as the only Area Path is selected by default. If you need to send a comment to the reviewer(s) you can enter this in the final textbox. As you can see, there are 3 modified files for review. If you are wondering which files are modified, you can click the **View Changes** link. Also, you can add related work items for this review in case the reviewer(s) need to check them.



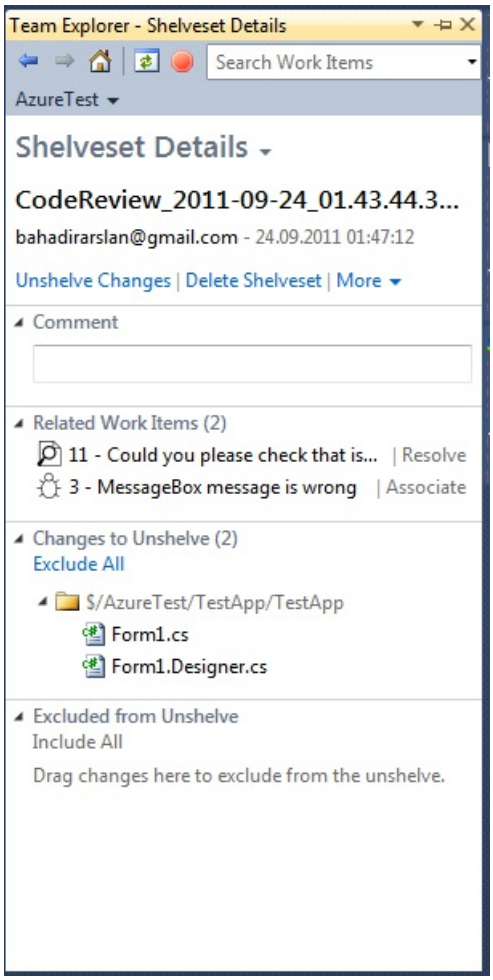
When everything is completed, click the **Submit Request** button. After you submit your request, Visual Studio 11 creates a workflow by shelving your pending changes. When the reviewer opens his/her **My Work** view, your request is displayed as you can in the image below.



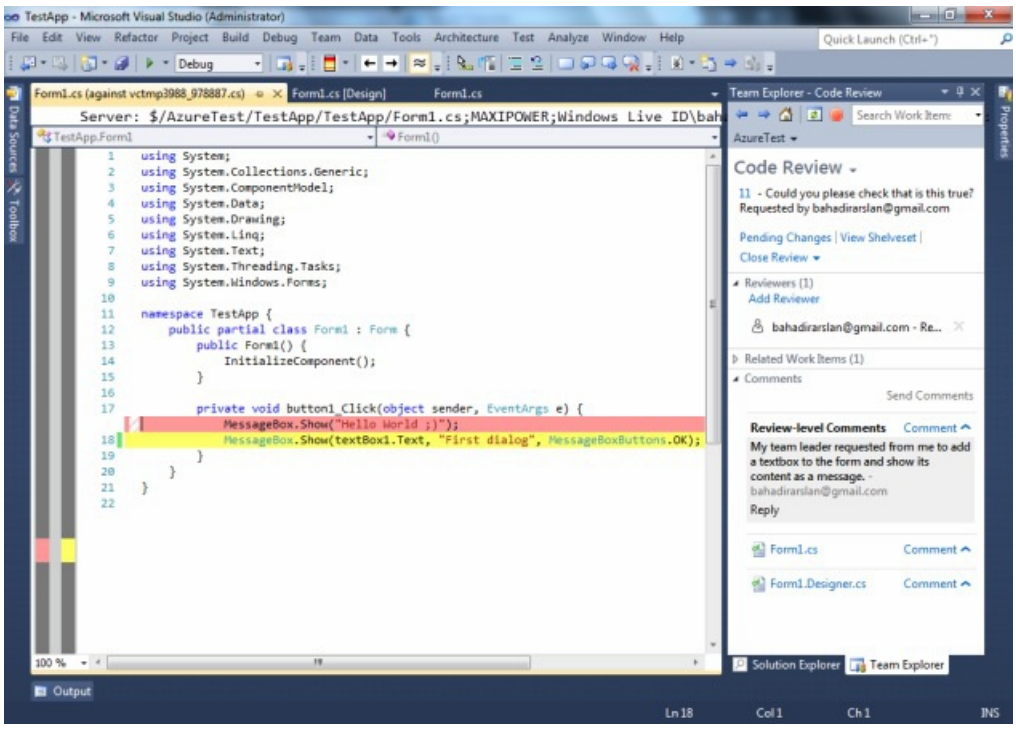
As you can see above, a new section had been added to the **My Work** view named **Code Reviews & Requests**. All of the incoming and outgoing requests are listed here; however you can filter by incoming requests or outgoing requests. If you want, you can see your requests from **Work Item Query.** You have to double-click a request to open it.



All details of requests are shown like the one above. What is the title of the request? Who had requested it? There is a list of reviewers, related work items, comments and files had sent to review. If you want to see more details about the shelveset or wish to unshelve the shelveset you can click the **View Shelveset** link.



To review codes, you can click **files**. When you click, Visual Studio opens the file within the compare mode, so you can see what developer had changed.

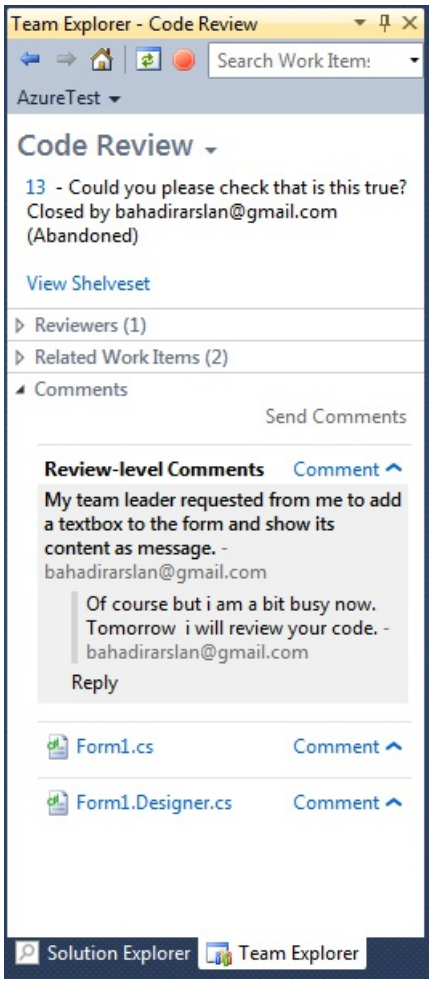


If you have any comments, you can click the **Comment** link beside the file name and then enter your comment into the opened textbox.

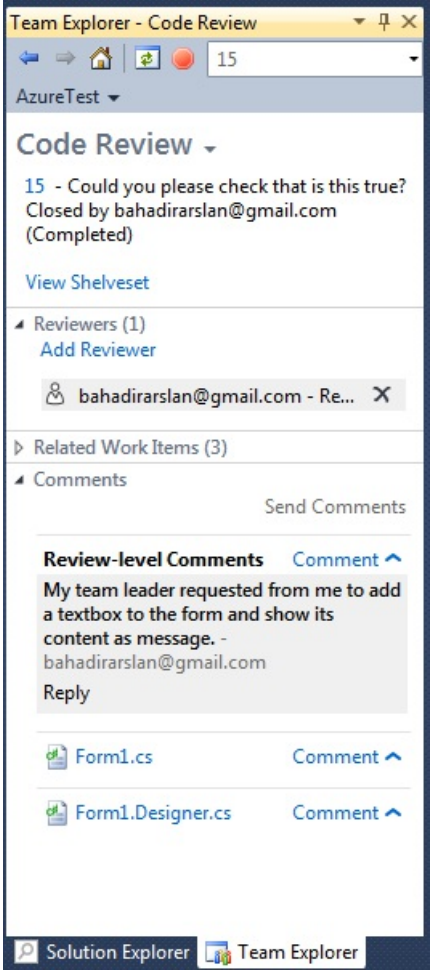
Maybe you want to reply to the developer's comment? You can click the **Reply** link below the comment.

After you complete the review, you should click the **Close Review** link at the bottom of the **Code Review** section. Now you have two options, **Abandon** or **Complete**.

Let's turn back to the developer's perspective. After the review is completed, you can see details at the **Code Review -> Request work item**. All code review requests create a work item.



The screen shot above shows the details of the abandoned review. The screen shot below shows the completed review's details.



As you can see, the **Code Review Workflow** has been designed to be as useful as possible to the average development team.

Conclusion

I have tried to describe the new features of Visual Studio 11 and the TFS Service. In my opinion, even in the current version, the TFS 11 Developer Preview seems to be excellent, and it will only continue to improve in subsequent versions. I was particularly impressed by the new Team Explorer and was almost as excited by the Code Review Workflow too. I checked out the Test Manager 11 too, but I couldn't find any major differences, so decided against describing it in this article.

I am planning to write another article that focuses more on Team Foundation Server 11, so look out for my new article because I saved several details for use later.

If want to download and try Visual Studio 11 Developer Preview for yourself click here and download.

> Bahadir has also written a review of the providers of Hosted Team Foundation Server 2010