

## "Server-side Caching" or "Buy speed with cache"

Jimmy Nilsson, JNSK

### Introduction

Even after fine-tuning your database, your ADO code, your network, and so on, you will eventually come to the realization that there is another way that beats them all when it comes to improving performance – caching. Caching is an old technique for increasing performance by storing frequently used data, therefore reducing costly round trips. Another possibility is the reduction of resource usage of less often used components in your system. The trade-off is less strain on memory for increased design and coding complexity (if the data is not read-only). Scalability and efficiency are factors that play a major part in the lives of developers. With the introduction of the .NET initiative from Microsoft, these factors will become more important, since the services found on the net have to be more and more scalable every day!

This paper will start by discussing some of the basics of caching. The focus will then shift to techniques for caching data in web-applications, both in ASP and in COM+. We will end with some case studies and recommendations.



#### Jimmy Nilsson

Jimmy Nilsson is the owner of the Swedish consultant company JNSK AB ([www.jnsk.se](http://www.jnsk.se)). He has been working with system development for almost twelve years (with VB since version 1.0) and, in recent years, he has specialized in component-based development, mostly in the Microsoft environment. He has also been developing and presenting courses in database design, object-oriented design, etc. at a Swedish University for six years. Contact: [Jimmy.Nilsson@jnsk.se](mailto:Jimmy.Nilsson@jnsk.se).

---

# Basics about Caching

## *Caching – Why?*

The goal with caching is not only to get better response times, but also to conserve resources from "hot spots" such as the database server or the network. It's often easier (and cheaper) to buy more memory than to buy more CPU cycles.

Let's take a metaphor from our ordinary lives. Every time you want to have a beer, you don't want to go to the supermarket. Instead, you have a cache in your refrigerator at home, right? Caching in a nutshell!

A simple programming example: imagine a loop where every iteration through the loop executes a function call that gives the same answer each time. A better approach would of course be to execute the function once before entering the loop and store the value in a variable. In the loop you can use the variable instead. This example is really little more than just good practice, and so I will focus on the bigger picture from now on.

## *Locality*

Caching is certainly nothing new. It has been used a lot, for example in hardware such as CPUs, disk controllers, and so on. The same goes for system software such as SQL Server and IIS. It is a common recommendation to monitor the buffer cache hit ratio (SQLServer:Buffer Manager) for your SQL Server. The rule of thumb says that if it's less than 90-95 percent, you probably have too little RAM and SQL Server has to hit the disk far too often. The result of that is obvious – namely, much slower response time, since disk I/O is much slower than RAM I/O.

Another example of caching in action occurs when an optimizing compiler tries to cache results in its registers.

When it comes to your application, there are several places where a cache could be introduced. You can have a shared one on the application level where all requests reuse the same cache. Another example is to cache data for each session so that every user only has to execute a specific function once. Watch out for scalability issues if you use ASP.Session for storing cache data. Each user will have its own instance and therefore you will eat up the server resources quite fast if you have a lot of users. You will also have problems with web-farms, since ASP.Session variables are pinned to a certain machine. There is a solution to that problem in ASP+. In ASP+ you can force the session-variables to be stored in SQL Server, but then ASP.Session is not a good solution for your caching purposes.

A more scalable place for storing your session cache is at the client side, but that is quite a long discussion and out of the scope of this paper.

## *Parameters to Consider Regarding the Data*

There are a lot of parameters to evaluate before you decide what to cache and where. If the *size* of the data to cache is very large, but the user only needs a specific and unpredictable part of the data each time, then you should perhaps forget caching and go to the database each time instead. It is possible to have a shared cache too, but a user-specific cache is probably not a good idea, unless you can put the cache on the client, in which case it isn't such a problem, as server-side memory won't be further eaten up by each successive new user.

Examples of other parameters to consider:

Is the data *read-only*, or must it be *updateable*? If it must be updateable, then must it be *real-time consistent*? If the answer to the last question is "yes", then it's typically best to skip

caching (especially in a server-farm) and go to the database each time instead. You lose much of the advantage if you have to keep going to the database to keep the cache current, and the code for doing this can be very complex and result in performance degradation. On the other hand, if the cached data is read-only, then it's probably better off cached.

Microsoft has observed that a lot of developers frequently start using caching without evaluating if it's actually needed. This often results in lower performance than would have been achieved anyway, without the caching! (See the *Resources* section.) In my case studies (below), you will see that you can benefit a lot from careful caching. There are big differences between different approaches, and certain requirements definitely mean that you would be better off without a cache. Remember, there is no such a thing as a free lunch!

### ***Units of Caching***

Another tricky decision is deciding what the cached unit should be. As always, there are a lot of different choices to make. You could choose to cache simple values like integers, strings, and so on. You could tag a string with custom tags, for example, "key=value;nextkey=nextvalue;". You could also use units with built-in metadata such as ADO recordsets and XML documents.

A slightly different solution is to use an object as the unit. Then you can have both the data (totally hidden) and the logic, if there is an algorithm for finding the data needed in the cache.

### ***Data as Constant HTML in ASP***

It's possible to work with static data, presented in HTML, but I'm not recommending it. When it comes to data that changes every now and then, you should definitely not use this approach, because you have to make many alterations whenever a value changes, which proves to be tedious and error-prone. You need to use a cache that is simple to set up and use, otherwise you will inevitably end up writing complex code simply to use and maintain it. Transformed data solves some of these problems.

### ***Transformed Data***

It is often highly beneficial to cache transformed and finished data as HTML in ASP applications for direct usage, for example on a web page. That way you don't have to use any more CPU cycles once the data is fetched from the cache.

There are, of course, drawbacks with this approach. (I'm not a salesman so I see the problems too) If a user wants to personalize his or her view of the data, then a shared cache with transformed data as HTML is probably impossible. This is one reason why the .NET Framework is based on XML – we can cache the data as an XML string and then transform it with XSLT to the final HTML, another method of achieving the transformation that will prove to be more robust than the above method.

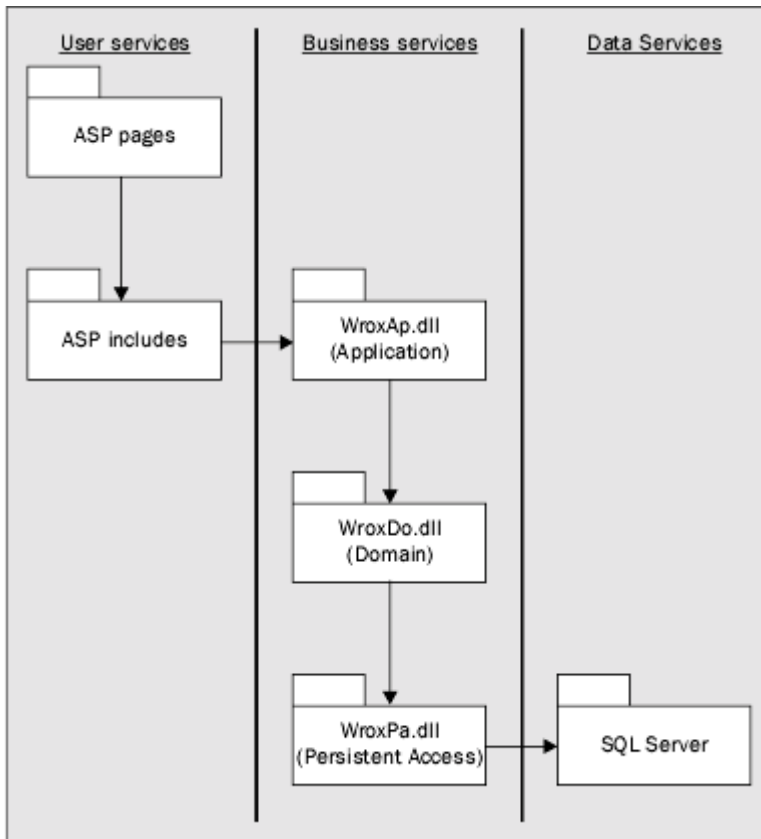
The diversification between different client types will probably continue to grow at it's current pace – HTML and DHTML are currently most common, but use of WAP with WML is on the increase every day. ASP+ controls will output what is needed for the particular client. That way you don't have to worry about whether the client is IE, Netscape, or something else, although this won't output valid WML – at least not yet.

Another problem is that the size of transformed data is typically larger than that of raw data. This is apparent if you compare a vector with ten elements and an HTML table with the same elements. You will consume less memory with raw data, which can be very important, making it possible for you to cache more.

Finally, I'm not fond of a high dependency on ASP (especially script-only). ASP+ seems to solve a lot of the problems I see in that environment, but even so, I will probably try to put as much code as possible (except for presentation code) in COM+ components (or their future equivalent).

## Caching in a Typical Architecture

Let's take a look at the architecture that I typically use for building web applications. The model is arranged in an unusual manner for space reasons – typically the architecture is shown with all the layers beside each other.



Before I discuss where caching can be used in this architecture, I will briefly describe the layers:

- ASP includes – All interaction between the ASP pages and the Application layer passes functions in ASP includes. One reason for this is to share code, but also to make it invisible to the ASP pages whether caching is used or not.
- Application (for example WroxAp.dll, the classes will be labelled by `ap`) – Use case centric. Each use case has a class here. Simple interfaces and use case-specific.
- Domain (for example WroxDo.dll, the classes will be labelled by `do`) – Concepts like order and customer are put here. The main layer for business rules together with the database. Generalized interfaces.
- Persistent Access (for example WroxPa.dll, the classes will be labelled by `pt` if they are transactional, or `pn` if they are not) – All database interaction is put here. There is one class for fetching customer information and one class for writing customer information, enabling transactions to be supported by one class and not the other, if desired. Generalized interfaces.

- Database – Typically a lot of usage of stored procedures and also a terrific place for taking care of data-centric business rules.

If you have read the Duwamish Online sample by Microsoft (see the *Resources* section), or something similar, you probably recognize the architecture used in this paper, even though different layer names are used, so I'm not claiming that I invented it! I did however produce the naming used here, which gives a good sort order for the projects in a Visual Basic (VB) project group. For example, WroxAp, WroxDo and WroxPa.

**Note:** the proposed architecture from the latest Duwamish version for .NET Framework-apps is very similar to what I use now. One difference is that Duwamish only use the Domain Layer for writes and let the Application layer directly call Persistent Access for reading data from the database.

Application, Domain and Persistent Access are typically implemented as configured COM+ components. I quite often move ASP includes to an ASP helper layer in COM+ instead – mostly because I like the VB environment better, but it's also in preparation for the arrival of WebForms. It's important to point out that none of the layers in COM+ know anything about ASP or HTML (except if an ASP helper layer is introduced), and ASP pages and includes don't know anything about COM+ or databases either. Encapsulation is the main keyword in my designs.

Caching typically takes place in the following layers:

- ASP includes
- Application
- Domain
- Persistent Access

Caching takes place in SQL server too, but that functionality is performed by the server itself. You can also use caching in the ASP pages for local variables and such, but that is not what I'm going to focus on here. Architecture is a huge topic in itself, which I don't want to get bogged down in. The purpose of showing the architecture now is to help you understand the case studies later on.

## Different Managers

There are a lot of different managers that provide solutions to caching problems, not all of which are discussed here. Examples that are *not* discussed are the Global Interface Table in Win32, the Registry, Global variables in COM+ components (see the *Resources* section), and Active Directory.

### *ASP.Application*

If you have worked with ASP, you have probably tried the intrinsic ASP Application object. It's a name/value dictionary where you can store values and objects. Watch out for storing apartment-threaded objects there. Normally you can't do it but there are workarounds. However, your performance will suffer!

If you need to protect several writes to the Application object to make them atomic, you can use a call to `Application.Lock` followed by a call to `Application.Unlock`. Observe, though, that the lock is for the complete Application object and, if you need to use it a lot, you will have high contention there (users will be forced to wait for other users' sessions to finish). This contention will slow things down – yet another potential performance hit caused by your effort to improve performance.

Example code can be found in the code for the case studies.

## ***Commerce.Dictionary***

Another name/value dictionary is found in Site Server Commerce Edition. I have played with that component but wasn't satisfied with the performance and the lack of information. As a result, I haven't used that component in the case studies for this paper. I do of course recommend you to test it and decide on your own opinion.

One neat thing with this component is a syntactical goodie. Take a look at the code below:

```
Public Function ListboxCustomerGet() As String
    ListboxCustomerGet = gobjDictionary.customers
End Sub
```

The `customers` property of the `Commerce.Dictionary` instance (`gobjDictionary`) isn't really a property but a key to the value.

## ***LookupTable***

The ASP team observed developer's problems with the Dictionary in the Microsoft Scripting Runtime. There were crashes because of threading issues, so they built a new but unsupported dictionary called `LookupTable`.

The `LookupTable` is simple to use. One small quirk is that the values must be loaded from a file. Even though `ASP.Application` is itself a dictionary, it's quite common to put a `LookupTable` instance in `ASP.Application`. This automatically gives a simple implementation for an "aging view" and gives smaller locks (at least if you have several instances of `LookupTable`).

Although the component isn't supported, it *is* delivered with the source code. Of course, using unsupported code can have unpredictable results in different situations. There's no guarantee it will always work. (See *Resources* for where to find more information and the download.)

Example code can be found in the code for the case studies.

## ***Files***

Using files for managing a cache is a bit different from the other managers I discuss here because, in this case, the data isn't held in RAM. (You save RAM, but reading from disk is of course much slower than reading from RAM.) Besides simple values, you can also serialize objects to files and de-serialize as necessary.

There are a lot of different approaches to taking care of files. Below, I show how it can be done with the native file handling in VB6.

```
Public Function ListboxCustomerGet() As String
    Dim intFile As Integer
    Dim strBuffer As String

    intFile = FreeFile
    Open "c:\temp\customers.txt" For Input As intFile
    Line Input #intFile, strBuffer
    Close intFile

    ListboxCustomerGet = strBuffer
End Function
```

Another common and more modern solution is to use the File System Object in Microsoft Scripting Runtime. Here is an example.

```
Public Function ListboxCustomerGet() As String
    'Add a reference to Microsoft Scripting Runtime

    Dim fso As FileSystemObject
    Dim tst As TextStream
    Dim strBuffer As String

    Set fso = New FileSystemObject
    Set tst = fso.OpenTextFile("c:\temp\customers.txt")
    strBuffer = tst.ReadLine
    tst.Close
    Set tst = Nothing
    Set fso = Nothing

    ListboxCustomerGet = strBuffer
End Function
```

### ***SPM (Shared Properties Manager)***

The SPM is a process wide area for storing global data/objects to configured components in COM+/MTS. (Watch out for storing VB6 objects because of their thread affinity.) Values are organized in property groups and then stored as properties. There are some parameters to set when you create a new property group regarding lifecycle and locking. Otherwise, the SPM is very easy to use.

A good source for more information about the SPM (and much more) is Ted Pattison's book *"Programming Distributed Applications with COM+ and VB6"*. (See the *Resources* section.)

Example code can be found in the code for the case studies.

### ***Extra SQL Server (on the Same Machine as IIS)***

I guess that SQL Server doesn't need any introduction, except for a few words about the context. If you decide to use SQL Server for caching purposes it can be a good idea to pin the tables to RAM. You can also have SQL Server execute the most commonly used stored procedures on startup so that they will be cached, and update the back-end through the cache server, by using triggers or doing the update through a stored procedure. Replication is another tasteful solution to the update problem. If you're going to run both IIS and SQL Server on one machine, then it needs to be powerful one, as both are very resource-intensive.

When it comes to locking support, although expensive, SQL Server still beats all the other managers discussed here by far. It gives you, for example, different isolation levels to choose from, such as read committed and serializable. There is also a difference between shared and exclusive locks.

Another scenario for using extra SQL Server-installations is for getting a persistent cache in front of a mainframe.

Example code can be found in the stored procedure code for the case studies. The code is not specifically written for an extra SQL Server-solution, but can be used in that scenario too.

## ***Custom Caching (ATL Component)***

A good solution for caching is to put the data and the logic together in a custom component. By building that component with ATL you get a very lightweight component with the correct threading model for agility and for being a good citizen in ASP.Application.

If you're not familiar with creating ATL components, you must be prepared to spend some time to get up to speed. See the comments below in the section called *Implementation Comments for Case Study 2*.

Example code can be found in the code for the case studies.

## ***ADO.Recordset***

The recordset is a powerful manager for holding data, with filtering possibilities and metadata retention. Even so, both Troy Cambra (at Microsoft) and Ted Pattison (see *Resources*) recommend that it shouldn't be used when sharing between different clients.

During the course of my experimentations, I found that I couldn't make it work when I tried to save a disconnected, client side recordset to ASP.Application in IIS5 (even though I used `makfre15.bat` first). I got an error message saying that I was trying to save an apartment style object to ASP.Application. In IIS4 it works fine, but IIS5 is a bit stricter about what counts as an agile object. In the case studies, I used a connected recordset, which works, but isn't a good solution in my opinion since then the data will actually be fetched from the database server each time you use the recordset. For more information, see the comments below in the section called *Implementation Comments for Case Study 1*.

Example code can be found in the code for the case studies.

## ***Scrrun.Dictionary***

Finally, I would like to discuss one more dictionary. It's the one found in Microsoft Scripting Runtime. It started to become very popular in the ASP community, but after a while, developers noticed crashes when using it in the ASP.Application. The object was marked as both threaded in the registry, but it wasn't thread safe, which led to trouble. (See Robert Carter in *Resources* for more information.)

My recommendation is not to use it for storing in ASP.Application. There are better alternatives!

## ***Managers – Summary***

Let's summarize the discussed managers concerning farm enabling and reference holding.

When it comes to farm enabling, you can, in most cases, build in support for this on your own. The same goes for preparing for the "web-gardens" of the future – the same application running as several processes in the same machine. Farm enabling refers to when you want to cache data that changes sometimes, but you want to have several instances of a shared cache, pinned to the different machines in the farm. If the real-time requirement is "real-enough", then you can use "aging views". The cache is then refreshed at certain time intervals, and the different caches won't be out of sync for more than the time interval. The LookupTable has built-in support for that concept if you use `LoadValuesEx` instead of `LoadValues` to load the data from the file.

Only ASP.Application, SPM, Files, and SQL Server are free from the need to be "held" by another manager. Some objects, however, are dependant of these objects themselves, for example a LookupTable object must be "held" by ASP.Application

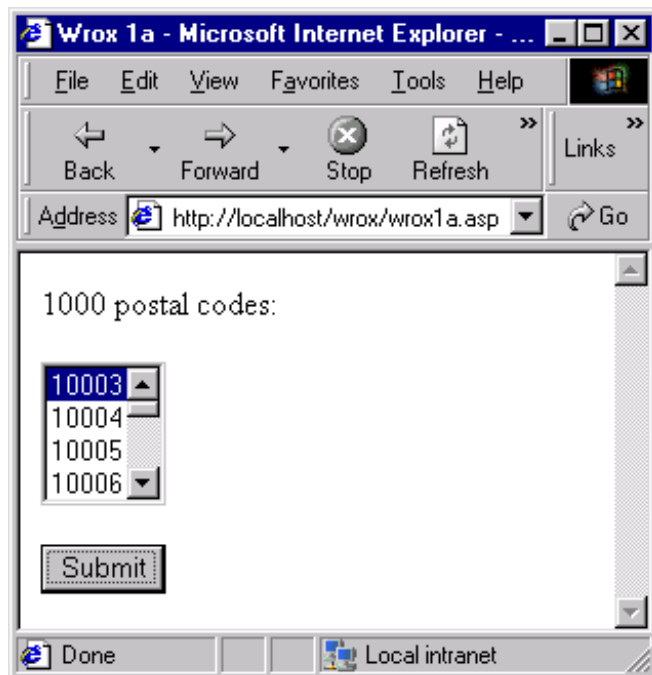
## Case Studies

Now it's time for the case studies. I have chosen to use two extremely simple case studies. The following database will be used:



There are four rows in the countrypart table and 1000 rows in the postalcode table. The model describes a situation where an organization has decided to split the country into four parts (north, south, east and west) depending on the postal code (zip code). If you know a certain postal code, the part of the country containing that postal code area can be determined.

The first case study will create a page displaying a listbox with all the postal codes to choose among. The second case study will choose a postal code and get the part of the country for that postal code. This is shown in the following two screenshots.





Before I go into the details, I will take you through some early design and deployment decisions:

- All database interaction via stored procedures. Using stored procedures is typically a faster access mechanism than using direct SQL via ADO.
- I used sloppy code, especially regarding error trapping, because it was designed to be a fast test. The same goes for implementation of the ObjectControl-interface, user-defined interfaces, and so on. I said in the Architecture section above that I like to generalize the interfaces in the Domain Layer and the Persistent Access layer with user-defined interfaces, but I haven't done so in these simple tests.
- I used the COM+ Application as a Server application. If you use the Library application instead you will get better results for some of the tests, but you will lose in fault tolerance.
- I ran IIS with low application protection (IIS process).
- `global.asa` was not used. This makes debugging a little easier, but at each request a test is needed to see that the cache is in place. By using `global.asa`, some of the tests will be faster.

### ***WCAT (Web Capacity Analysis Toolkit)***

There are a bunch of different testing tools for end-to-end testing of web applications. The one that I used is called WCAT, a free application created by Microsoft (see *Resources*). It's quite raw, but it does a good job, is easy to set up, and proves to be a very powerful tool when you need it to be! This was the first time I used WCAT, and it only took an hour or so to get up and running.

You can install WCAT so that you get a directory called `server`, with a file called `server.exe`. I chose not to unzip `server.exe` with all the data files to simulate transaction mixes. Instead, I used a very simple script that only asked for the same page over and over again. Typically, you will want to mix different transactions and different input for each request, so that you get a more realistic test than the one I used here. In my simple tests, SQL Server will have everything needed for the next request in its cache.

I asked the controller to execute with one client, five client threads and no think-time. I used 60 seconds for testing, 10 seconds for warm-up and 10 seconds for cool-down. You will find all the configuration files and scripts that I used together in the source code for this paper.

## ***Test Environment***

The three machines used for the test environment had the following specifications:

- 1: Combined client and controller – PII 266, 128 MB, NT4 (sp4)
- 2: Web and COM+ server – 2 x 533, 256 MB, Windows 2000 AS, IIS5
- 3: Database server – PII 650, 256 MB, Windows 2000 Server, SQL Server 7 (sp2)

All machines were connected to a 100 MB network. It's switched but not loaded at all, except for my tests.

## **Case Study 1**

As I said previously, this test is to present a listbox with 1000 postal codes.

### ***Compared Solutions for Case Study 1***

Test 1a – the following tests were performed:

#### ***No cache, go to the database each time.***

Clean, simple and flexible. A stored procedure is used in the database and `GetString` is used on the client side recordset. In the code fragment below, `GetString` will serialize the recordset as a string. I have programmed it to add `</option><option>` tags between each row. This results in a superfluous `<option>` tag at the end of the string, but it's easy enough to strip it off afterwards. Syntax example:

```
strTmp = "<option>" & rst.GetString( , , , "</option><option>" )
```

#### ***Recordset cached in ASP.Application.***

The recordset can now be used over and over again – for example, with `GetString` – to create an HTML-formatted listbox.

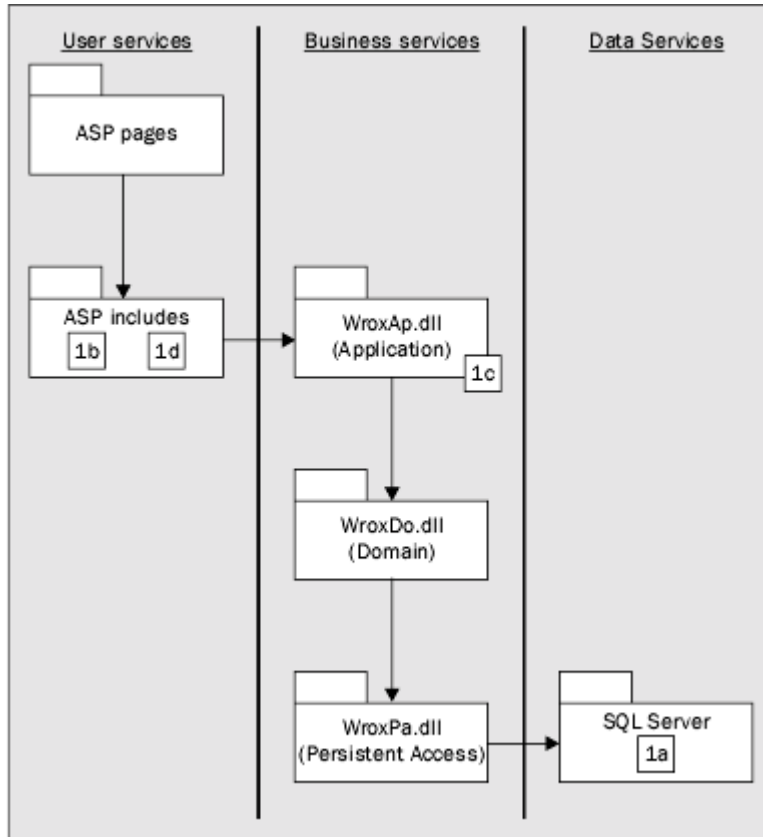
#### ***Customized string cached in the SPM, used from the Application layer.***

The recordset is transformed to a custom tagged string with `GetString`. That string is then put in the SPM. The string is easily transformed to valid HTML with a single `Replace` statement in ASP includes.

#### ***Transformed HTML string for the listbox cached in ASP.Application.***

The string can now be used to directly render the listbox at the HTML page.

## Architecture for Case Study 1

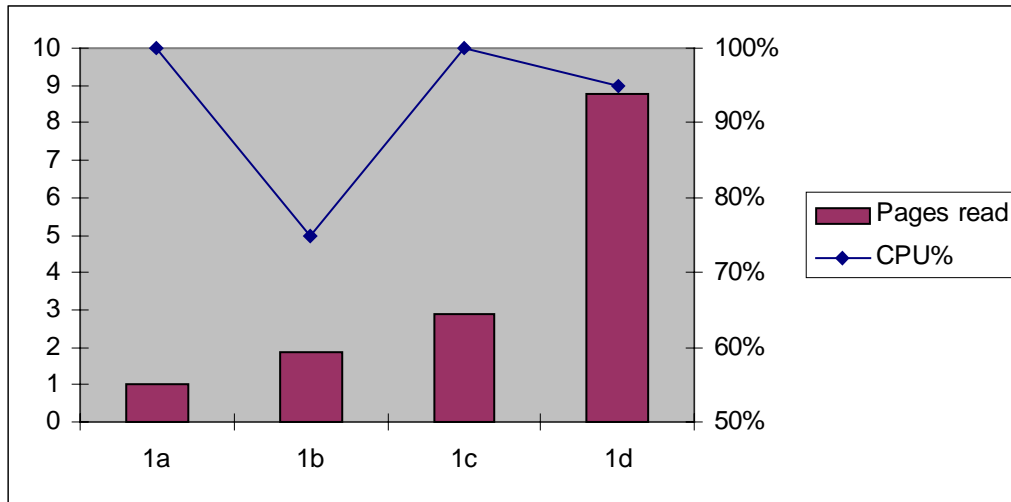


Both tests 1b and 1d use the ASP.Application object as a manager. 1c uses a LookupTable instance held by the SPM, hidden in the Application layer. 1a is not using caching at all, except for that performed by SQL Server behind the scenes.

### Performance Comparison for Case Study 1

I gave test 1a the number 1 and then recalculated relative numbers to the other tests for the number of pages read. Test 1b has 2, 1c has 3 and test 1d has almost 9. 9 for 1d means that test 1d is 9 times faster than test 1a. Observe that the comparison is for end-to-end. If we isolate the different caching solutions and only compare them without communication overhead, etc., the difference would be much larger!

The CPU-usage shown in the result below is for the IIS/COM+ server.



It's worth mentioning that test 1b also steals approximately 50% CPU usage from the database server. (Test 1a takes less than 10% CPU usage from the database server. 1c and 1d take 0%.)

### ***Implementation Comments for Case Study 1***

As I have said before, the solution for test 1a used a stored procedure to do the `select` from the `postalcode` table. `GetString` was used on the recordset to get a HTML-formatted string.

The implementation for test 1b didn't go as expected. As mentioned in the *ADO.Recordset* section, I didn't manage to put a disconnected, client side recordset in `ASP.Application` for IIS5. Instead I used a connected recordset, which means that the database will be contacted each time the recordset is used. I also raised the `CacheSize` to 1000. Also note that, with IIS4, I couldn't make it accept the recordset delivered from my VB component, but it worked if I had all the code in ASP. My intention was to cache the data too, but ADO/IIS didn't allow this.

For test 1c, I used `GetString` on the recordset to get a custom tagged string in the Application layer. I didn't want to use an HTML string because I didn't want the application layer to have any presentation dependencies. The string is then put in the SPM, and is easily transformed to valid HTML with a single `Replace` statement in ASP includes.

Finally, the implementation for test 1d is very simple. Just use 1a to get an HTML string and put that in `ASP.Application`. Next time, fetch the string from `ASP.Application` – voilà!

Be sure to inspect the code to get a closer look!

## **Case study 2**

This case study aims to find the part of the country that a specific postal code lies in.

### ***Compared solutions for case study 2***

The following tests were performed:

#### ***No cache, go to the database each time.***

This time only one row is delivered from a stored procedure, so a recordset isn't created - instead I'm using an output parameter. All the logic is put in the stored procedure.

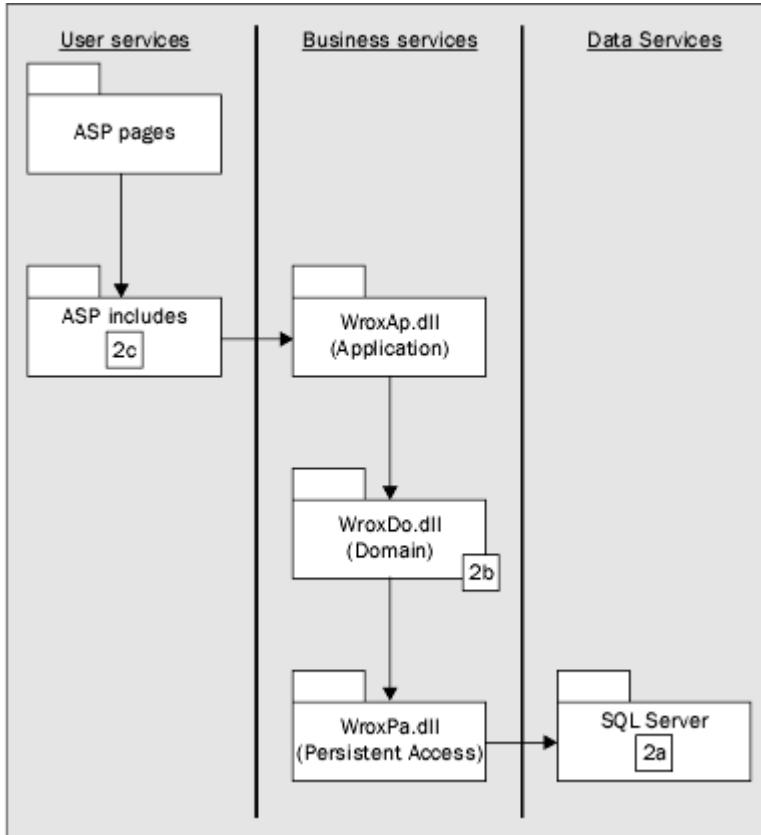
### ***A LookupTable hidden in the Domain layer.***

The LookupTable is stored in the SPM, and retrieved from there each time. It could be cached in a global variable too, to skip the overhead with the SPM, but I believe that any resulting difference in performance is quite small.

### ***Custom cache as an ATL component in ASP.Application.***

When I discussed this paper with Troy Cambra at Microsoft, he proposed that I use an ATL component for this case study. That solution puts the data and the algorithm together in a very lightweight component. I'm mostly a VB and SQL guy so I hadn't used ATL before. More comments about this will follow.

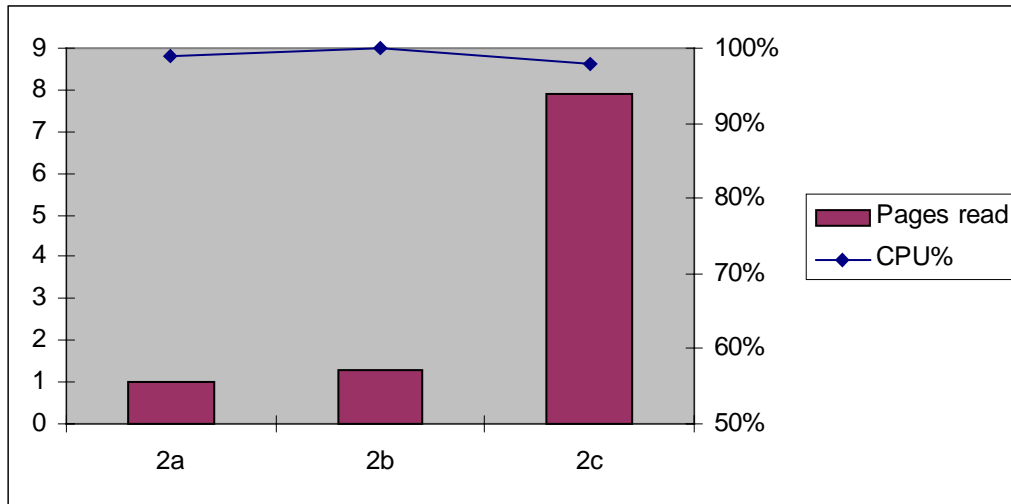
## ***Architecture for Case Study 2***



2c is using an object stored in ASP.Application, hidden by ASP includes. 2b uses a LookupTable stored in the SPM in the Domain layer. 2a is not using caching at all, except for that performed by SQL Server behind the scenes.

## ***Performance Comparison for Case Study 2***

I gave test 2a the number 1 and then I recalculated relative numbers to the other tests for the number of pages read. Test 2b became approximately 1.5 and test 2c got an 8, which means that test 2c delivers 8 times the number of pages delivered by test 2a. The CPU-usage shown in the result below is for the IIS/COM+ server.



2c was CPU bound on the client and there were a couple of connect errors (maybe five). I could, of course, configure more than one client, but since it's CPU bound on the server too, I probably wouldn't get a much better result.

I tried a `POST` of an HTML form for the second test in WCAT, but the results were poor. When I tried sending the postal code as a parameter in the querystring, I got the results shown above.

### ***Implementation Comments for Case Study 2***

I have already mentioned that, for 2a, I used a stored procedure that returned the part of the country as an output parameter. That solution is quite straightforward. If the given postal code is not found, a default value is given from the stored procedure.

The solution for 2b uses an instance of `LookupTable`. The instance is shared between all the requests and to accomplish that, it has been stored in the SPM. At the first request, the instance is not found in the SPM and therefore all the postal codes and their respective part of country are read from the recordset with `GetString`, and written to a file called `lookup.txt`. This file is put in `c:\temp`, a place where it can't cause conflict problems (or any others, for that matter). When running the COM+ application as server or library). `lookup.txt` is then loaded into the `LookupTable` with `LoadValues`. Finally, when the value is to be extracted from the `LookupTable`, a check is done with `KeyExist` to see if the postal code could be found. If not, a default value is given, otherwise a call to `LookupValue` is used.

The solution for 2c uses an ATL component stored in `ASP.Application`. When I discussed this paper with Troy Cambra at Microsoft, he said that using an ATL component here would be a terrific solution. I hadn't worked with ATL before, but when I started it seemed simple. I used a VC++ 6 wizard and made some choices – both for threading model, dual interface, Free Threaded Marshaler, no aggregation and `ISupportErrorInfo`.

I added an interface with three methods: `SetDefaultRegion`, `Add`, and `GetRegionFromPostalCode`. `SetDefaultRegion` is used for setting a region (part of country) to use when a postal code is given that is not found in the database. `Add` is used at the setup of the object, to add each postal code/part of country pair. Finally, `GetRegionFromPostalCode` is used to translate a postal code to a part of country.

I was told to use an STL map for storing and retrieving the information, but when I started to write the code, I soon realized that it wasn't easy, and got frustrated when I had several strange problems with such a tiny little component. I read bits from a few of Richard Grimes'

books (see *Resources*), followed by an article in MSJ by Don Box (see *Resources*), and finally asked my friend Dan Byström at Visual Design Softscape AB for help. He told me that I can't use Swedish settings in the Control Panel if I want to build an ATL project! It's *not* easy to find that information, trust me!

He ended up writing the STL code for me and set a Gx-switch to escape an exception handling problem – different in STL to the equivalent in the default VC++ configuration. Finally, before making the project compilable as a minimal executable, he had to delete the preprocessor directive `ATL_MIN_CRT` so as to not receive the error that `Main` was missing. (The debug-version compiled just fine before that.) All this for 25 extra lines to the wizard generated code! And to top it all off, the string handling is messy too!

Hopefully I'll get some time to play with ATL again. It is very efficient when you actually get it working! I would ideally prefer to use Visual Studio.NET with VB.NET, as they seem to be able to create agile objects. C# (C sharp) will probably also be a good solution for me and all the other developers who haven't got up to speed with ATL yet.

Once again, be sure to inspect the code to get a closer look.

## Final Comments

### *Updating?*

In the case studies, I expected read-only data so no attention has been paid to the updating problem. If you want to cache updateable data in a farm enabled architecture (or garden enabled, as will be possible in ASP+), then you can, for example, implement an "aging view" strategy as I mentioned earlier (note that aging views are not real-time consistent).

If you need to update a file for use by `LookupTable`, for example, it could be written to by the stored procedure that does the update to the database. The update could also be made from a trigger and from the update code in `COM+`. The trigger will execute even if the update is written directly to the database, an advantage in some scenarios.

If you don't need to have a farm enabled architecture, then it's easy to fix real-time consistency too, but you don't usually have the cache in a transacted manager, so take care. If you really need real-time consistency, my tip is to skip caching and let SQL Server do the job as usual. This will normally give good performance anyway. Essentially, the more updating you need to do, the less you're likely to gain from caching – and you might even cause performance to decline.

A good rule of thumb is to "think big". Even if you don't think that you need a farm today, perhaps you will tomorrow. The whole industry is going in the direction of needing more and more scalable solutions!

### *What's to Come Soon?*

ASP+ will have something called Output Cache built in. Use the following directive in the beginning of a page:

```
<%@OutputCache Duration="15"%>
```

If exactly the same querystring is requested twice within 15 seconds, the page is not regenerated.

At first this solution may seem like all you need, but remember that each type of client will need a specific output-cached page (if the page being cached is ready to render for the browser, such is the case with HTML). The following scenario could also present a problem:

- List customers (output cache created, 60 seconds)
- Add customer
- List customers again - this step has only taken 30 seconds to reach

Hopefully Microsoft will fix things so that a regular post of a form could be used too, and not only querystring based requests. As I understand it, it will also be possible to cache fragments for output, not just complete pages.

ASP+ will also have what is called the ASP+ Cache. You can use it in much the same way as you use ASP.Application. One advantage of ASP+ Cache is that you can set up dependencies for when the cache should be deleted. A typical dependency could be time, a change in a file, or memory usage combined with request rate.

At the time of writing, I didn't know very much about ADO+. Hopefully, the DataSet concept will be a good solution without all the problems found with Recordsets. Time will tell.

### ***What's Missing from this Paper?***

By using MSMQ/Queued Components, you can get a different type of cache than those I've been focusing on here – a write cache. When you have written a message to the queue, you can forget about it and go on to do other stuff. The message at the queue will then be processed when appropriate.

As you have probably noticed, I haven't made great mention of XML in this paper – this is because I didn't want my focus to be too broad as there would have been a danger of straying away from the main subject matter. Caching XML documents is a very good solution and if you can do it then it's great, especially when done on the client side.

Another reason for using XML documents is if the data has a non-trivial structure. Most of the techniques discussed in this article are different dictionary objects, which work best with simple data like key/value-pairs.

Intel's 64-bit processor Itanium and Win64 will soon be giving us the possibility to use terabytes instead of gigabytes of RAM. I have a good idea of how to use that extra memory!

Microsoft will soon release what they call Internet Security and Acceleration Server 2000 (see *Resources*). One of its services is caching.

Quite late in the beta cycle for Windows 2000, Microsoft withdrew a service called IMDB (In Memory DataBase). Perhaps we will see it in an upcoming version (or something similar).

### ***Recommendations***

In my opinion, it's very important to encapsulate the caching decisions you make. Everything that you don't show to the surroundings can be changed easily and with less risk of side effects and the introduction of bugs.

Don't trust any published test results (not even those in this document!). Perform your own tests, in your own environment and with your own applications.

Finally, don't cache if it's not necessary. Check with a prototype before you build. Remember that you introduce complexity with caching!

*Special thanks to Troy Cambra at Microsoft and Dan Byström at Visual Design Softscape AB.*

## Resources

Richard Anderson, Alex Homer, Rob Howard, Dave Sussman; A Preview of Active Server Pages+; Wrox; 2000; ISBN 1-861004-75-3

Don Box; The Active Template Library Makes Building Compact COM Objects a Joy; <http://www.microsoft.com/msj/0697/ATL.htm>

Robert Carter; Abridging the Dictionary Object: The ASP Team Creates a Lookup-Table Object; <http://msdn.microsoft.com/workshop/management/planning/MSDNchronicles2.asp>

Duamish Online, Sample project from Microsoft; search at [msdn.microsoft.com](http://msdn.microsoft.com) for more information

Richard Grimes et al; Beginning ATL 3 COM Programming; Wrox; 1999; ISBN 1-861001-20-7

Richard Grimes; Professional ATL COM Programming; Wrox; 1998; ISBN 1-861001-40-1

Internet Security and Acceleration Server 2000; [www.microsoft.com/isaserver/prodinfo/overview.asp](http://www.microsoft.com/isaserver/prodinfo/overview.asp)

Edward A. Jezierski; COM+ Application Guidelines for Visual Basic Development; <http://msdn.microsoft.com/library/techart/complus.htm>

LookupTable component; <http://msdn.microsoft.com/workshop/server/downloads/lkuptbl.asp>

MSDN Show; [msdn.microsoft.com/theshow/Episode003/Transcript.asp](http://msdn.microsoft.com/theshow/Episode003/Transcript.asp)

Jimmy Nilsson; MTS (and COM+) and global variables; <http://www.vb2themax.com/HtmlDoc.asp?Table=Articles&ID=100>

Ted Pattison; Programming Distributed Applications with COM+ and Visual Basic 6.0, second edition; Microsoft Press; 2000; ISBN 0-7356-1010-X

Sundblad and Sundblad; Designing for Scalability with Microsoft Windows DNA; Microsoft Press; 2000; ISBN 0-7356-0968-3

Microsoft Visual Studio Next Generation; [msdn.microsoft.com/vstudio/nextgen](http://msdn.microsoft.com/vstudio/nextgen)

WCAT; [msdn.microsoft.com/workshop/server/toolbox/wcat.asp](http://msdn.microsoft.com/workshop/server/toolbox/wcat.asp)