Chapter 1
Database Driven Web Systems – The Big Picture

This chapter provides an overview of the elements that comprise interactive, database driven web sites. Systems that integrate both web and database technologies have begun to dominate the web. They are part of the world most of us now live in – we search for news, buy books, music and other items and increasingly use the internet as a learning environment. We'll take a look at the various pieces involved in these systems and then try to make more sense of things by following a transaction across the system. Pay close attention to how the various elements relate to each other. By the end of this course, you will have created a system that integrates web and database technologies. It's pretty cool and not that difficult. Starting with a broad overview of the technologies will make the creation of your own system much more understandable. It might seem overwhelming at first, but we'll be taking it one step at a time.

The broad Overview

Patrick is a dedicated paintball player. As the weekends approach he starts chatting about the cost of CO_2, his new auto-hopper and the location of his favorite paintball fields. He and his friend Michael have decided to blend their interest in paintball with some newly acquired computer skills to create a paintball database and to make that database available to others through their web site. The database stores product information in various categories … markers (i.e. guns), protective masks, paintballs in dozens of colors and camouflage clothing. It also stores the location, price, ranking and directions for paintball fields. These fields are never found in the Yellow Pages. You learn about the fields through word of mouth or from flyers taped to the walls of paintball shops. Patrick and Michael hope their web site will provide another avenue for learning about the paintball fields. From a drop-down list on the web page, visitors select a region and the system returns a page listing all the paintball fields in that area! It's actually pretty slick. Do you know of a new field? Well, why not fill out a form on the web page and add that paintball field to the database.

These two guys have integrated a number of technologies into a pretty useful system. On a different level, they've also managed to create a community – paintball fans will soon use this site to comment on various fields, share their opinions of the latest equipment and possibly use a newly developed web page to join paintball teams. The interactivity of this system is what people now expect of a computer

system. The technologies involved are the technologies that drive the modern world of commerce. You've experienced this in the form of internet stores, but it has also developed as a critical element in business to business transactions and government sites that allow you to register your car, browse the census data and many other tasks.

While the user sees this world from the simplistic view of a web page, there is a complex dance taking place between databases stored on computers (they could be anywhere in the world), computer code that places data into and retrieves data from the database, the internet as a means of transferring things between the database and the user sitting at home – and of course web technologies that allow descriptions of web pages to travel the internet and to be reconstructed by a web browser.

Is it complicated to build one of these interactive systems? Well, I guess the answer is maybe. If you want to develop a commercial web site that provides shopping carts, a variety of payment methods (including credit cards) and a number of shipping options – then it's time to hire the professionals. On the other hand, developing a system similar to the paintball system we just described is well within your grasp. Before we get to the details of how these systems are built, I think it's helpful to introduce the various elements and provide a sense of how they relate to each other.

May I serve you? The basics of clients and servers

In every database driven web site there is a clear division of labor. Some technologies are sitting on a computer that houses the database and web pages, while others are sitting on your local machine. The computer hosting the web site and database is classified as a **server** because it "serves up" the web pages and database data to some **client** sitting at another computer. In some sense, *you* are the client – but more precisely, some piece of software sitting on your computer is the client being served by that "web server".

Your client machine has a piece of software called a web browser. You probably recognize that browser by the name "Internet Explorer" or "Mozilla Firefox". It's the job of the web browser to request web pages from web servers and to build those pages once they arrive. There are strict rules (computer people call them **protocols**) for this process of requesting and sending web page descriptions. Both the client and server adhere to a standard protocol called **Hypertext Transfer Protocol (HTTP)**. You've seen evidence of this whenever you have browsed the web. Even if you get sloppy and type www.cnn.com to reach CNN, your browser quickly points out that communication with that web server will need to use the set of rules set forth in HTTP. The full entry will be corrected to http://www.cnn.com.

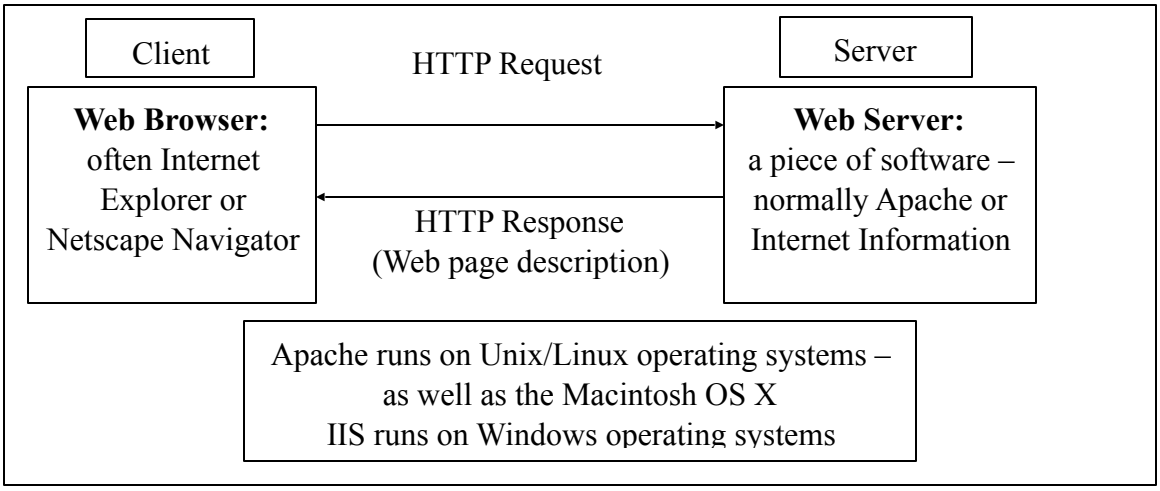


Figure 2. The role of clients and servers in web activity

Web page Descriptions: Only the story travels…

Notice in Figure 2 that the web server only sends you a description of the page. That description includes content as well as formatting instructions. When a web page is described, a special language is used. **Hypertext Markup Language (HTML)** provides a standard way for web developers to describe the look of each page. HTML has codes to describe the background color of the page, its text color, the placement of images and other text – as well as providing links to other web pages, both within this web site and to other sites. Figure 3 has a simple HTML page for you to check out. Each HTML code is called a **tag** and can be recognized and decoded by a browser. This description is the only version of the web page that the web server sends. It will be up to your browser to read this description and build a page that matches it. In the process, the browser will need to make a number of further requests to the web server – each image, for example, will have to be requested as the browser needs them. There is an amazing flurry of activity between the client and server as the page gets built. Each request for the next piece and the transmission of that piece takes place following the HTTP protocol that we described a few moments ago. Check out the special section on Network Sniffers at the end of this chapter – you'll get some appreciation for the interaction between the browser and web server.

```
<html>
<head>
    <title> Intro to Sailing </title>
</head>
<body  bgcolor= "#000203">
   <center>
     An Introduction to Sailing
     <img  src= "SailboatPicture.jpg" >
   </center>
</body>
</html>
```

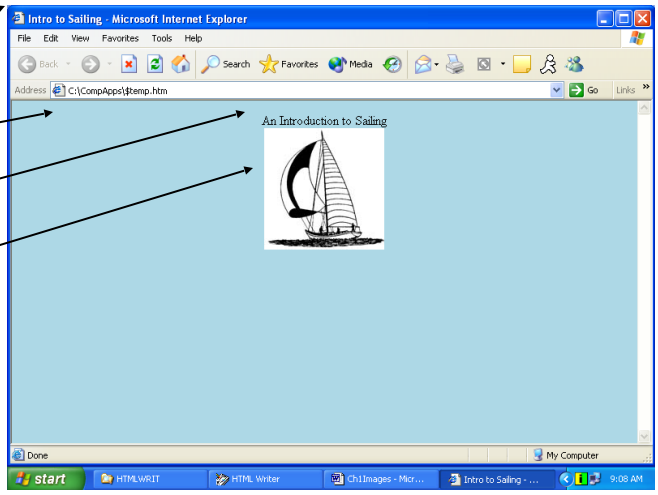


Figure 3. A browser builds web pages based on descriptions in HTML code

And now the database…

At this point we have a client and server involved in sending and receiving HTML documents over the internet – each using the rules of the HTTP protocol. What's missing is the connection to the database. In a small system, that database is probably sitting on the same machine that houses the web server. In a large commercial operation, the web server and database may each be housed on their own computers and connected to each other by a local area network (LAN).

The database houses a collection of data that can be queried – that is, you can ask it questions. You can also put new data into the database, modify and delete data. As you might have guessed by now, there are standards for everything – including a standard language for talking to a database! That language is **Structured Query Language (SQL).** You can pronounce SQL as 'sequel' or just say the letters SQL. It can be very English-like. Here's a simple SQL statement that gets all the data from a table of data called Customers, provided the customer lives in Rhode Island. The asterisk is just a wildcard that grabs all the data about each customer. By the way, computer people call the asterisk 'star' – just to be different! They would say "Select star from Customers where the state is Rhode Island".

```
SELECT * FROM Customers WHERE State = 'RI'
```

You should be wondering at this point how the client manages to ask the database a question. Sure, SQL will do the asking…but how do you get to the SQL? That process will involve a special kind of HTML (a **Web Form**) and something sitting on the web

server that understands that some action needs to take place (**ASP**). That action often includes establishing a connection to the database and executing a SQL statement. Let's start with the Web Forms**.**

Web Forms –

Web forms are nothing new. Every time you buy something online you fill out a form that provides the seller with your name, shipping and billing address and credit card number – that's a web form. They can be fairly sophisticated, with radio buttons to select shipping methods, drop-down lists for state names, sometimes even a comment area for you to provide special delivery instructions. Whatever elements they contain, they all include a submit button of some type. The button might say 'order now' or 'sale is complete' or some related message – but that button is classified in HTML as a submit button and every web form has one. Each choice you make on that page is stored in a variable that gets passed to some page that's sitting on the web server… and that's where things start to get a bit crazy. You see, your web form is told to pass its data to an ASP page that is sitting on the web server – basically a HTML page with some special coding on it. It is this *other* page that will do all the talking to the database. Once you hit the submit button, the data is passed over the internet to the ASP page on the web server. That page has a SQL statement on it that allows communication with the database (Figure 4).

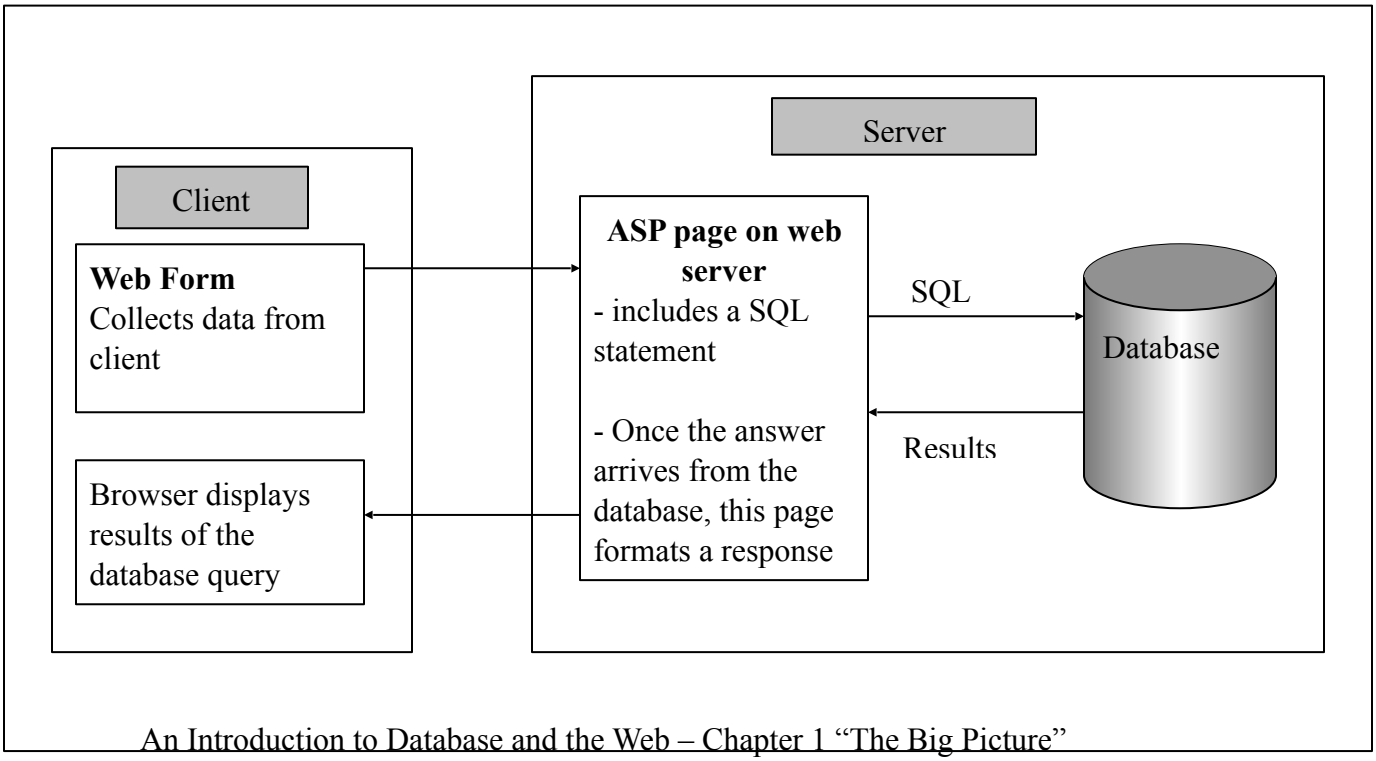


Figure 4. A web form passes data to an ASP page that interacts with the database

ASP (Active Server Pages)

Active Server Pages are the final piece of this system. ASP is actually a process that runs on the web server – think about it as a program that is always ready to run when data from a web form arrives. Its job starts when it recognizes ASP code on that special web page that just got handed data by the web form. It is ASP that makes sure the SQL statement gets run and that the results coming from the database get formatted in a way that can be sent back to the client. In the paintball system, the web form might have provided a way for the user to select Southern Ohio for their paintball field search. That selection was stored in a variable and passed to some ASP page sitting on the web server. The ASP page made sure a SQL statement was executed to search the database for paintball fields in Southern Ohio. When the results arrive from the database, that ASP page uses the results to build a web page listing all the Southern Ohio fields. That page then gets sent back to the user! Thankfully, the user never sees the ASP code – which would be pretty confusing; complete with SQL statements, loops, connection strings that provide info to the servers about the database we are connecting to… along with lots of computer logic (stuff like IF statements). Much of that computer code is written in a programming language such as VBscript or JavaScript that browsers and servers can interpret. All the user sees is the list of paintball fields they were looking for. If the user were to examine the HTML code, all of the ASP would be gone – replaced by the results that came from the database.

Following a transaction

Let's try to pull these ideas together by following a customer who is looking for a book to purchase. Pay attention to the role of client and server processes.

NileBooks.com is a massive online book store. On the powerful servers they own, the company has created a database that stores data related to books they have for sale. The database stores the ISBN number, title, author, price, cover photo, description and category (mystery, biography, historical fiction etc.). Let's just deal with that much for this example. In an upcoming chapter you'll learn how the various reviews, inventory levels, suppliers and other data can be related in a more complete system. The company has also created a web site using IIS (Internet Information System) as the web server. Remember, that's the web server running on a Microsoft Windows machine. If this were

a UNIX machine, they would be running Apache as the web server. You live in a world where Microsoft Windows is the dominant desktop operating system, so you'll be surprised to find that most web servers are actually running on UNIX or Linux machines (see Figure 5). There are many web servers out there, but Apache and IIS dominate the scene.

Most popular Web Servers (as of June 2008) Source: *Netcraft Web Server Survey*		
Web Server	**Market Share**	**Number of sites**
Apache	50%	86,845.000
IIS	35%	62,400,000
Others	15%	

Figure 5. Apache and IIS are the dominant web servers

For simplicity, let's focus on two of their web pages. They have a web form with a drop-down list that let's you select a category of book – mystery etc. Their second web page is actually an ASP page that can send SQL statements to the book database. The web form will determine the book category to search for and the ASP page will run a SQL statement to find those books.

Very little activity appears to be taking place on the client side. The user begins the process by starting their web browser – Internet Explorer or Netscape Navigator. They have purchased from NileBooks many times and have the URL (Uniform Resource Locator) memorized. They enter NileBooks.com in the browser's address bar and press Enter. As always, the browser corrects them and enters http://www.NileBooks.com. At that point the browser sends a request for the home page at NileBooks. It formats the request according to the HTTP standard. For a browser, this is how you talk to a web server. The web server at NileBooks accepts the request and prepares to pump out a description of its main web page.

That home page happens to be a special web page – a web form that includes a drop-down listing the various categories of books that you can search through. It also has a submit button, but they've replaced the word "submit" with "Begin your Search". This page is packaged according to the rules for an HTTP Response and handed to the Internet for delivery to the browser that asked for the page.

As the page arrives at your machine, the browser gets busy. What has arrived is a description of what NileBooks wants the page to look like. The browser opens the description which is written in HTML. The HTML code indicates that the background color needs to be a dark blue and that an image called "OurLogo.jpg" needs to be in the top right hand corner. The browser quickly formats another HTTP Request, this time for

the image it needs. NileBooks accepts that request and quickly sends the image as part of a HTTP Response package.

Piece by piece the page takes form. In the early days of phone modems, speeds were slow enough for you to follow the browser's progress as the page was built; one image arrived, then slowly the next one trickled into place, a table was drawn and the background color adjusted. It could be a frustrating process – but you could see what was happening. With today's high speed connections, you're given the illusion that pages arrive fully formed (Check out "Just for Geeks" at the end of this chapter).

The user now has NileBooks' main page on their screen. They click the drop-down and select 'Mystery' from the list. At the bottom of the page there's a "Begin your Search" button – they give it a click. The user expects to get a list full of mystery books, probably sorted by publication date with the most recent books listed first.

We know what kind of book they want, but who do we tell? As it happens, the web form includes the name of an ASP page sitting back at the NileBooks web server. You won't see this written on the page the user is viewing, but it is definitely there. Clicking the "Begin Your Search" button tells the browser to send the results of the web form back to the web server where the data will be handed to that ASP page. There isn't a whole lot going back, just the value of the drop-down list (mystery) and the fact that you clicked the submit button. As always, the data gets packaged as HTTP traffic and is handed over to the internet for the trip back to the web server.

The web server is running ASP and recognizes the incoming traffic as something that needs to be given to a particular ASP page. That page makes a connection to the Books database and prepares to execute a SQL statement. How the word "Mystery" gets included in the SQL is beyond our story for now, so just accept that it somehow gets into the statement. Notice how the SQL asks for the results to be sorted by publication date with the *biggest* (most recent) dates listed first. To go in reverse alphabetical order or from high numbers to low ones, you say the order is *descending*.

```
SELECT * FROM Books WHERE Category = 'Mystery' ORDER BY PubDate DESC
```

The Books database performs the query and retrieves each book in the Mystery category. Our SQL requests that the list be sorted with the most recent books at the top of the list. The database makes the collection of mystery books available to the ASP page that asked for it.

Now is not the time to get into the details of how the next step happens, but the ASP page opens the list provided by the database and begins to format it as if it were creating a new

web page. That page includes the title of a book along with author, price and description. The name of the image holding the cover photo is included as well. This process is repeated for each book in the list. Once completed, this web page is sent back to the browser where it will be the browser's responsibility to build the page that matches the HTML description (review the process in Figure 4).

I know this seems like a complicated process – but much of it is handled by the browser and web server. We'll take our time and guide you step-by-step as you build your own system. Our first step is to develop the database. You'll learn to design tables of data and to relate them to other tables. You'll learn to create queries that answer some fairly sophisticated questions. Those query skills will eventually translate into an introductory knowledge of SQL, which will provide your web sites with a window into the database. Once the database is up and running, it's time to design and code the HTML that describes your web pages. The final weeks of the course will integrate the database and web technologies. It's at that point that we'll poke around in ASP – the glue that connects the web with your database.

Keep in mind that your system has the potential to be extremely powerful. The paintball system we discussed at the start of this chapter was a neat idea; it also went far beyond the mechanics of creating web pages and databases. We spoke of developing a community. Do you have a hobby you'd like to share with others, perhaps a sport or even an interest in politics? As you think of a semester long project, keep in mind that your database has the potential of being a two way street – not only can strangers search it, but they can add their own special knowledge to it.

Will you know everything about database driven web sites? Well of course not. The complexities of real commercial web sites are far beyond this book. A great deal of the story we've presented in this chapter is also greatly simplified. As an example, the internet doesn't understand URLs like www.NileBooks.com. The internet only understands addresses called *IP addresses*. They look like 192.168.230.129. We haven't talked about those addresses or how and when a URL gets replaced by one. We also have ignored the whole complexity of how the internet carries your traffic and how it 'knows' how to find the destination. Just where is www.NileBooks.com? Want to see some of these details? Check out the *Just for Geeks* section at the end of this chapter.

As you might have guessed, some of these issues will be dealt with as we move along. Hopefully this chapter has begun to place the important technologies in context. You should understand the concept of client and server and the fact that it represents a division of labor that has much of the activity happening at the server. I hope you also appreciate the role of the browser in requesting web pages in a standard format called HTTP. Keep in mind that the browser builds the web page based on a simple text description sent by the web server. For now, do your best to follow the database, ASP

and SQL connections. This is fairly complex on the first pass. We'll come back to these ideas in later chapters where some of the questions you might have will be answered.

Just for Geeks – Watching Web Traffic
Sniffing the Network!

People who maintain computer networks sometimes troubleshoot problems with a tool called a ***sniffer***. A sniffer is a tool for eavesdropping on traffic (data) as it moves along the network. We can use a sniffer to take a fascinating peek into the activity that accompanies the arrival of a web page at your browser.

Before we start the "sniffer", let's think about the activity we expect to see. The browser will need some way to find the web site – where is CNN? Once the browser knows where to go, it needs to ask for the home page and then make additional requests for objects such as images. The details in the sniffer "capture" are beyond this book – but you might find some of the activity interesting as a web page arrives at your machine.

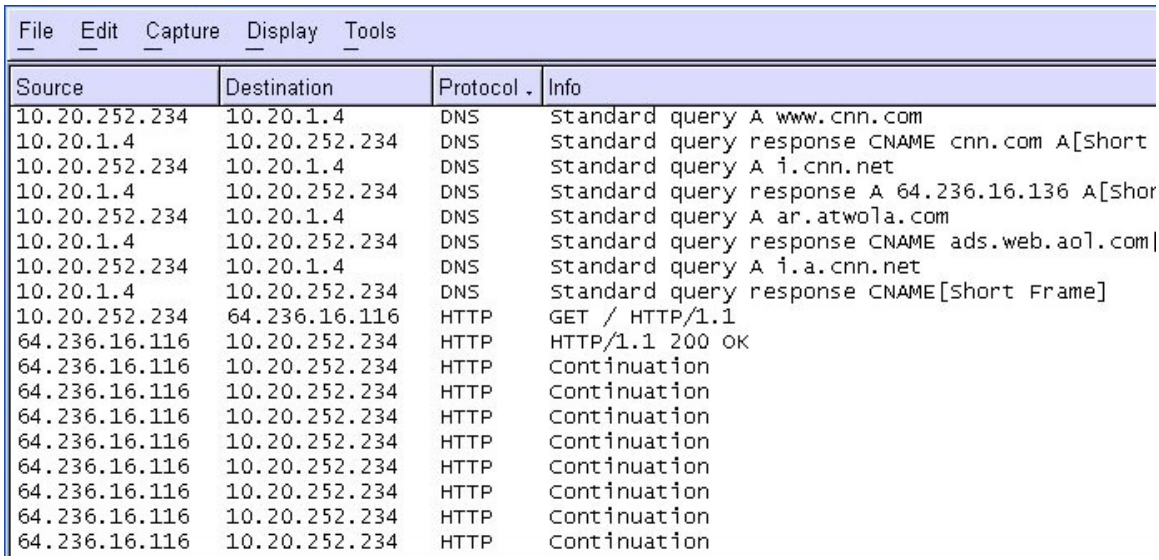
The process of getting a web page begins when a person enters a URL into the address bar of their browser; such as www.cnn.com. That address is a ***domain name*** and (from the computer's view) is nearly worthless. Rather than domain names, the internet is based on addresses called ***IP addresses***. Each IP address is composed of four numbers separated by dots – like 192.168.25.7. The web browser will need to ask someone to look up www.cnn.com and determine the corresponding IP address. Who handles that job? Well, there are special servers that handle Domain Name Services (DNS). You can see from the following figure that there is a DNS request ("what is the IP address of www.cnn.com?") followed by a DNS reply – that reply includes the IP address we were looking for. There are a number of requests and replies in the sniffer capture we made. Focus on the first two and don't get hung up on the details here – just be aware that your browser needs to request the IP address of CNN. Think of it as trying to call someone with just their name – the phone won't work until we look that name up and find the corresponding telephone number.

Once the DNS activity dies down, the browser uses HTTP to ***GET*** CNN's main page. That page will be broken into sections that are small enough to travel across the internet (the small packages are called ***packets***). Notice how the request to GET a page is followed by a sequence of packets marked as "continuation" – in other words, the page was too big to send as one packet, so here are a series of packets that are a *continuation* of what I 'm trying to send you.

From the following sniffer capture, can you figure out what my IP address might be? (Hint: what IP is the *source* of the first message?)

DNS is looking up the IP address for us – can you find the IP address of the DNS server? (Hint: who did we send our DNS query to?)

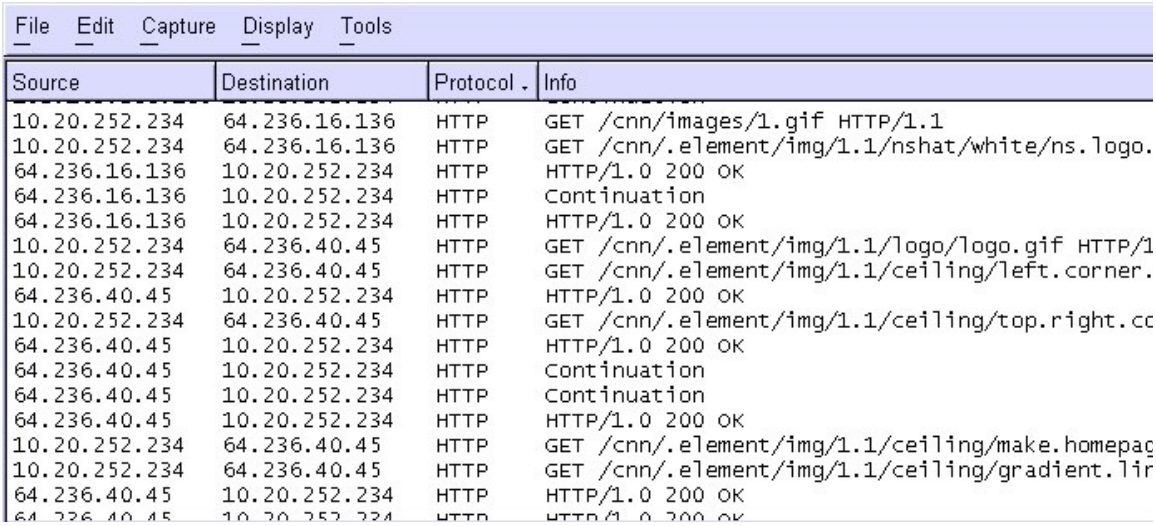
Can you figure out what CNN's IP address is? (Hint: once I finish



File Edit Capture Display Tools

Source	Destination	Protocol	Info
10.20.252.234	10.20.1.4	DNS	Standard query A www.cnn.com
10.20.1.4	10.20.252.234	DNS	Standard query response CNAME cnn.com A[Short
10.20.252.234	10.20.1.4	DNS	Standard query A i.cnn.net
10.20.1.4	10.20.252.234	DNS	Standard query response A 64.236.16.136 A[Shor
10.20.252.234	10.20.1.4	DNS	Standard query A ar.atwola.com
10.20.1.4	10.20.252.234	DNS	Standard query response CNAME ads.web.aol.com
10.20.252.234	10.20.1.4	DNS	Standard query A i.a.cnn.net
10.20.1.4	10.20.252.234	DNS	Standard query response CNAME[Short Frame]
10.20.252.234	64.236.16.116	HTTP	GET / HTTP/1.1
64.236.16.116	10.20.252.234	HTTP	HTTP/1.1 200 OK
64.236.16.116	10.20.252.234	HTTP	Continuation
64.236.16.116	10.20.252.234	HTTP	Continuation
64.236.16.116	10.20.252.234	HTTP	Continuation
64.236.16.116	10.20.252.234	HTTP	Continuation
64.236.16.116	10.20.252.234	HTTP	Continuation
64.236.16.116	10.20.252.234	HTTP	Continuation
64.236.16.116	10.20.252.234	HTTP	Continuation
64.236.16.116	10.20.252.234	HTTP	Continuation

Here's a screenshot taken about 50 packets (packages) from what we just looked at. Your browser is still trying to gather the pieces it needs to complete the web page. At this point it is requesting some of the images that CNN wants on the page. Each GET you see is a request from the browser to the server – get me this, get me that. "Continuations" are just packages that were too big to travel without being broken into smaller units.



File Edit Capture Display Tools

Source	Destination	Protocol	Info
10.20.252.234	64.236.16.136	HTTP	GET /cnn/images/1.gif HTTP/1.1
10.20.252.234	64.236.16.136	HTTP	GET /cnn/.element/img/1.1/nshat/white/ns.logo.
64.236.16.136	10.20.252.234	HTTP	HTTP/1.0 200 OK
64.236.16.136	10.20.252.234	HTTP	Continuation
64.236.16.136	10.20.252.234	HTTP	HTTP/1.0 200 OK
10.20.252.234	64.236.40.45	HTTP	GET /cnn/.element/img/1.1/logo/logo.gif HTTP/1
10.20.252.234	64.236.40.45	HTTP	GET /cnn/.element/img/1.1/ceiling/left.corner.
64.236.40.45	10.20.252.234	HTTP	HTTP/1.0 200 OK
10.20.252.234	64.236.40.45	HTTP	GET /cnn/.element/img/1.1/ceiling/top.right.cc
64.236.40.45	10.20.252.234	HTTP	HTTP/1.0 200 OK
64.236.40.45	10.20.252.234	HTTP	Continuation
64.236.40.45	10.20.252.234	HTTP	Continuation
64.236.40.45	10.20.252.234	HTTP	HTTP/1.0 200 OK
10.20.252.234	64.236.40.45	HTTP	GET /cnn/.element/img/1.1/ceiling/make.homepag
10.20.252.234	64.236.40.45	HTTP	GET /cnn/.element/img/1.1/ceiling/gradient.lir
64.236.40.45	10.20.252.234	HTTP	HTTP/1.0 200 OK

Obviously you don't need to understand the details of this process. Just appreciate the flurry of activity that must take place whenever a web page seems to magically snap into view on your computer!