

Scripting, Databases and Weblogs

By Rudolf Ammann

Hello and welcome to today's session of the *Internet Technologies* course, which is about "Scripting, Databases and Weblogs".

During the tutorial in the second part of the session, which will be held at the usual venue, I will ask you to set up a weblog and do some posting on the Web in accordance with an assignment that Melissa has written up for you. She might even join us later today on the Web, but I can't give you any guarantees on that.

Before we get to the tutorial, I would like to fill in some background for you, here in the lecture, and talk about forms of Web publishing that are a bit more advanced than what you are learning in this course, and that might be something to look forward to in another course at the Department of Information Studies, if you're interested in getting more deeply into the matter. This lecture is about scripting and databases – and the effect these technologies have had on the preferred forms of publishing on the Web, such as, obviously, weblogs.

This lecture is especially aimed at overcoming the puzzlement that many of you might be experiencing as a consequence of taking this course: most of you will be familiar with social networking sites such as Facebook and Twitter (to name only the two currently most popular of the bunch), or you might be running a weblog yourself. If you are, you might find yourself wondering about the difference in experience between using your Facebook account on the one hand, and coding HTML on the other. Both activities are a use of Internet technologies, both activities are a form of Web publishing, but they're very different, so you might ask is there even a connection between the two? Does Facebook employ an army of coders who will type out the HTML and the CSS whenever somebody posts something to their Facebook account?

Obviously not.

I would like you to think of the difference between these two publishing modes in historical terms, as a matter of growth and evolution in the underlying technology. Technological change happens in layers, as one new set of technologies gets layered on top of existing technologies. What

you've been learning in this course, coding HTML by hand, represents the first, original layer of Web technology. Ten years ago, most casual Web users would have agreed that this is how the Web gets build: you write your HTML pages locally, on your computer, then upload them to the Web, and that's the way it is.

It turned out, though, that things have become more sophisticated since, and most casual Web users these days don't even recognise the hand-coding and uploading of HTML as a set of cultural practices that any sane person would engage in of their own accord.

So as another set of technologies has been layered on top of the old Web, there has been a shift towards a higher degree of automation and, with it, a greater degree of convenience. These newer technologies are deployed server-side, meaning that they run on Web servers without any hand-crafting of individual pages required on your local computer.

This additional layer of technology was first introduced and put to actual use as early as 1994, almost as soon as the Web first managed to attract a non-specialist audience, but it wasn't until around the year 2000 that it became a widely used practice under the name of "web applications". More recently, of course, the same thing has been hailed as "Web Two Dot Oh", a term that I've never been particularly fond of, that happily appears to be past its shelf life now, and that I'll be doing my best to avoid in this lecture.

In what follows, I would like to describe, in very basic terms, the difference between coding HTML by hand, and those more advanced server-side technologies that have come to determine people's experience of the Web today. This additional layer of technology consists, if you allow me to simplify a bit, of two main building blocks: databases and scripting. I'm going to take these two things in turn, illustrating them, as I go along, with a particular application, the blogging service Wordpress.com, which you'll be using in the tutorial part of today's session.

What you have just learned

// What you have just learned in this course is called hand-coding Web pages. It's a valuable skill to have and it's a nearly indispensable basis for understanding what happens on Web sites that use databases and scripting for automated page generation rather than hand-coding. Let me remind you of the process involved in coding pages by hand. When you hand-code, you do two things, you // create and upload.

It's a three-stage process, actually, illustrated in // this little graphic from bottom to top:

1. On your local computer, you type your source code in a text editor.
2. You then upload your files to the Internet – and
3. The files are then sitting up there in the cloud where people can request and view them in their browsers.

(The cloud as a metaphor for the Internet, by the way, is a very old conceit and is getting a new lease of life today as “cloud computing” is being advertised as the Next Big Thing.)

The // technologies involved in these three steps are:

1. HTML and perhaps certain auxiliary technologies such as JavaScript and CSS
2. FTP, the File Transfer Protocol: or, as the case may be, a more modern transfer protocol such as SSH that is more secure.
3. Once the the uploaded file is sitting pretty in the cloud, once it has, in other words, been transferred to the file system of a Web server, it becomes available to the Web-using public through the magic of HTTP, the Hypertext Transfer Protocol.

So much as a reminder – you knew this already. I’m not totally sure you’re too familiar yet with the process of uploading your files, which involves an // FTP client.

You are looking here at a typical, generic FTP client that has a //two-panel layout in which the // left panel shows you the file system on your local machine and the right panel shows you the file system on the remote server. That remote server is a Web server that runs on a physical machine anywhere in the world, and the act of // uploading an HTML file involves the act of copying it from the left to the right panel.

The tutorial you’ll be using later today consists on of a hand-coded page of XHTML that I typed up for you locally on this very computer and that I uploaded to a machine physically located in California.

You will need to keep this idea of a remote Web server in mind to understand what follows, because a Web server can do more than simply receive the files that you upload to it and send them down the wire in response to incoming HTTP requests.

A web server can also run programs that assemble Web pages from various bits and pieces, and it can can run a database management system that stores the bits and pieces that are required for this to happen.

In other words: remote servers are the place where //scripting and databases interact to form this newer layer of technology that goes beyond hand-coded HTML.

With this newer layer you no longer:

//create and upload

Instead you do something else: you
// Automate and post

I'll be illustrating the process with a top hat standing for the database and a magician's hand representing scripting.

Databases

I've been harping a bit on the notion that scripting and databases form a "newer layer of technology" on the Web. It bears pointing out, though, that neither databases nor scripting are breaking news. Databases entered computing in the nineteen-sixties, and the most widely used standard language in which you speak to databases even today was rolled out by IBM in 1974: // SQL, which stands for Structured Query Language, and which is still a very big deal today.

What is a database? For a definition, you might say that a database is simply a storage and retrieval device for digital data. You put your data in there and you pull them out again in interesting ways. In our little top hat and magician analogy, the bunny-rabbit needs to go into the top hat so that the magician can // pull it out again.

//Databases mostly use one particular data structure for organising data: the table, which means that databases have a fair degree of similarity with spreadsheets: in both databases and spreadsheets, a single entry occupies a table row and can have as many data cells as the table has pre-defined columns.

Let's apply this idea to our interest in weblogs and see how it compares with hand-coding. When you hand-code an HTML page, you create a document that uses the markup language to structure the data and you put things together so that they form a well-designed page. By contrast, when you automate page creation on a Web server, you break down the content to be published into pre-defined constituent bits and pieces, so you can pull them out and assemble them later on, at page-creation time.

So, lets see how Wordpress breaks down things in its database: Wordpress is open-source software, which means you can download it for free and take it apart to your heart's content: we can thus inspect the // overall database schema, where there is, amongst other things, one database table called, sensibly, // "posts", and when we go in there and look at the internal structure of the entries that are stored in this table, we'll find in the // column headers that a post has, left to right:

ID: an identifier of the post, which may never be displayed but which will allow the application to locate the post in the database

author: a post has an author name assigned to it, as the weblog might in fact be authored by more than one writer

date: obviously it has a posting date

title: obviously the title under which the post will be displayed

content: this is the actual body of the post

excerpt: think of this as the short form or the introduction to a post

tag: this is a category into which a post falls: a piece of meta-data that characterises the contents of the post.¹

So these are the things that sit in the top hat. What we now need is some magic – or business logic, if you prefer – that will pull them out and put them where we want them be in our HTML code.

We need a // scripting language.

Scripting languages

Think of scripting as a form of programming that isn't done in traditional heavy-weight programming languages such as C, but in any of a number of light-weight languages that, typically, are less formal than conventional programming languages, more similar to natural languages such as English, that are easier to learn, more forgiving, and that allow for much quicker changes to be made. Scripting languages are said to be more “high level” than the conventional “low level” programming languages: they save programmers the trouble of coding low-level routines, but the convenience of being easier on humans comes at the price of being somewhat tougher on computers, as scripting languages typically require more processing cycles than conventional programming languages and, as a consequence, are slower.

To put things into historical perspective again – scripting languages aren't // new either. The grand-daddy of the lot, LISP, which has a venerable history as the language of choice in artificial intelligence programming, was created as early as 1958. More recent scripting languages include Perl, created in 1987, Python, created in 1991, and then Ruby and PHP, both of which were created in 1995. All of the latter languages have been used extensively in scripting the Web – the ones that stand out in terms of popularity, are Perl and PHP. Ten years ago, most Web scripting was probably

¹ This data structure is somewhat simplified and does not literally correspond to the “posts” table.

done in Perl. More recently, // PHP has become the most widely used scripting language on the Web. Interestingly, the acronym PHP started out as “Personal Home Page”, but it’s now believed to stand for “pre-hypertext processor”, which nicely captures its primary use, which is to automate the generation of hypertext markup.

(It’s also the language that Wordpress is written in.)

You might ask: What does scripting do on the Web? Well, scripting languages are especially good at manipulating strings of characters, so you can use them to create scripts that say: “dip into the database, get me the ten most recent posts and put them right here on the front page, add their titles and timestamps, and make sure they’re sorted in reverse chronological order, most recent first, oldest last”.

So, think of Web applications such as Wordpress as a number of interacting scripts that run on a Web server and that work as a kind of publishing engine: they pull the bits and pieces from the database and put those bits and pieces through a templating system that assigns them their place on the page, and then, well, they assemble the finished page from the constituent parts they found in the database.

Once we have thus automated the process of HTML generation on the server, the act of posting becomes simple – and it’s exactly what you’re familiar with from Facebook etc. Open your browser, go to the site, click on the button that says “new post” (or whatever) and you’re then presented with a // textarea like this one in Wordpress. You enter your text into the textarea and hit “publish”. Let’s // get a different perspective on this as we turn this browser window around a bit. Your post then // goes into the database. And then // through the magic of scripting, your post gets pulled out of the database, is put through a template and – hey presto – here’s a shiny new web page!

You might, in other words, call the web page simply a highly specific // database view – a bit like a report generated from a spreadsheet.

You might also call it a // database interface – a surface that lets you interact with the contents of a database.

Interfaces

One effect that server-side technologies such as the database and scripting combination have, is that the notion of a “page”, which in printing technology has been with us for half a millennium as the immutable basic unit of content distribution, stops being all that immutable. Once the ink is on the paper surface, changes become virtually impossible. But with advanced online publishing, the data

can be pulled out of the database in a variety of interesting views, and if one view is not interesting enough, well, you can create other views. As a consequence, the “page” becomes less important. What becomes more important is the smallest individual piece of content – call it a “chunk”, call it a posting. The question becomes: how are we going to present these individual chunks? The answer, increasingly, has not been the page, but the string of chunks, something that might better be called a stream and that sometimes, using a term from broadcast media, has been termed a “channel”, even.

As you would expect, the technology has spawned its own interfaces – or “data views” – that weren’t possible or even conceivable under the print-based regime, and the weblog, considered from a purely technological perspective, is one such data view, is one particular interface to the database.

Here’s how the weblog interface works, and to emphasise its novelty, let me contrast it with the typical interface of an old-fashioned hand-coded “personal homepage”, which is perhaps more “intuitive” because it corresponds much more closely to an older technology, that of the printed book.

Under the print-based regime, when you have any larger body of text, you need to make this text accessible through // access structures such as an index at the back of a book and a table of contents at the start.

Ten years ago, there were still lots of Web sites that were structured by analogy to this: somebody put up a bunch of individually crafted pages, sorted them into different topics, created different folders on the Web server that represented the “sections” of a site, and then, of course, had a “home page” up front that started with the gleeful exclamation “Welcome to my homepage!”, followed by something along the lines of “Here’s the stuff that’s available on my site” and then you’d get a neatly classified table of contents, // perhaps in decimal classification, probably not in decimal classification, but decimal classification offers a neat way of thinking about this: it’s an abstraction hierarchy that maps onto the Web server’s directory structure: a page named *dogs.html* would be uploaded to the directory “My_hobbies” in much the same way that *dogs* would be a section of a chapter on *hobbies* in somebody’s printed autobiography.

What you are looking at here is a random decimal classification. If you’ve ever been asked to structure an academic paper, you know exactly what this is about.

By contrast, then, let’s look at the // weblog interface. In the weblog interface there is no access structure up front that mediates between you and the content: when you go to a weblog, you always have the most recent posting shoved right in your face: the organising principle here isn’t hierarchical order, it’s that the // most recent goes on top, where its visibility is highest, and it’s //

reverse chronological sorting for the less recent postings.

I've put together a little scale model that illustrates the process. When there's a new posting, it goes to the top of the page: // Post 1 (never mind about the copy – it isn't meant to mean anything in this model). When there's a newer posting, it goes on top of the older one: // Post 2. Post 3 goes on top of the other two, //Post 4 goes on top again, //Post 5 goes on top yet again. At some point you'll need to decide that you can't allow the page to go on forever, and you will need to drop the older entries. Which is what happens here in our model: when //Post 6 gets added, Post 1 drops off the page.

Is Post 1 lost now?

No, because in the weblog interface, a post doesn't just get // written to the front page; another copy of the same post also goes to another location somewhere else on the site, where it is supposed to be part of a // persistent archive.

So any post, after it's been databased, gets pulled out of the database twice: it gets posted to the top of the front page (to the left), where newer material will get posted on top, and where it will eventually drop off the page, and it also gets posted to a persistent archival location (to the right in the illustration). And, crucially, there's a hyperlink that connects the two: whenever you read a post on the front page, you can click on the title, or a link underneath, and it'll take you to the permanent archival record of the same posting. This link, in technical language, is known as a // *permalink*.

And what we have here is a typical database interface: it's the same posting pulled out of the top hat twice: once to satisfy the need for immediacy in the streaming of updates on the front page, once for archival purposes further back into the site.

This brings up another subject that we should touch upon at least briefly: tagging.

// Tags

You might say: "I prefer the old table of contents because that gives me a better overview of what's there." Fair enough: the weblog interface obviously privileges the recent at the expense of the larger view. The weblog interface doesn't encourage the larger view – not in the sense that a book is expected to embody a larger view with a beginning, a middle and an end. Weblogs are open-ended: they are a stream rather than a hierarchical structure.

Nevertheless, the database enables an alternative view of the archives through the use of // metadata. Metadata is data that is "meta", meaning "about": it's data about data, like when you stick a label on something to communicate what it is or where it's from. Such meta-data labels are called

“tags” in computer science, and once you have “tagged” the contents of your //database (like our Wordpress database here), you can then easily // retrieve and display all the items that share the same tag. So, in addition to the reverse-chronological stream up front and the permalink, tagging offers another interface, another data view of a database’s contents.

This, in fact is what we’ll be doing in the tutorial in the second part of the session, where we’ll be using a strategy that is being used at many conferences nowadays where conference-goers are encouraged to mark posts that pertain to the conference with the official tag issued by the conference organisers: relevant posts can then be aggregated from all across the web.

You’ll be // tagging your blog posts with “g0017” to mark that they belong to this course. The site we’re using has a page where all postings tagged as “g0017” are aggregated, and this will allow you to check in on what everyone else is writing.

Web feeds

Let me wrap up this lecture with an additional aspect of scripting, databases and weblogs that won’t be part of the tutorial but that is nevertheless an important development that is deeply enmeshed with the subject matter: // web feeds and syndication, which offer yet another data view, yet another interface to the database.

Cast your minds back to the part a few minutes ago when we looked at the front page of a weblog and found that it isn’t a representation of the content to be found on a site, not an access structure such as an index or a table of contents, but that it presents the content itself, in a stream of reverse chronologically sorted items.

Let’s focus on this stream for a second. Back in the olden days, if you wanted to follow, say, ten weblogs, you would need to visit these ten weblogs individually, on a daily basis, and check out what was new.

Some people found this inefficient and reasoned: “wouldn’t it be nice to separate these streams from the front pages of all these different weblogs and pull them together – in other words, to aggregate them – in one place?”

Indeed it would. So a new approach came about in which this stream of content came to be provided in an alternative format that needed to be machine readable. HTML was not sufficiently strict for this, so new file formats were created, starting in earnest in 1999. The first of these file formats that turned out to be of importance in the larger scheme of things was // RSS, which [now,

mostly] stands for Really Simple Syndication. (// here's a post marked up in RSS, which shows that RSS is not that much different from XHTML, as both languages are XML dialects, have lots of angle brackets all over the place, etc.). Many developers found RSS wanting, however, and by 2003 they were coming up with an improved file format for content syndication, which they named // Atom. (//again, a post marked up in Atom, and again, even a cursory glance will reveal that this obviously is XML).

RSS vs. Atom is now one of those holy wars that keep raging among technologists over the relative merits of competing technologies. You have probably noticed the // common web feed icon before, which stands for both RSS *and* Atom, so end users, to whom this holy war is (mostly) irrelevant, need not worry about any of the particulars underneath.

So, how does this work?

First of all, after you've put a chunk of content into the database, it gets // pulled out of the database and written to the places we've discussed: the stream of the most recent updates on the front page for one thing and a permanent archival location for another (leaving tagging aside for the moment). With syndication, however, the content also gets written to yet another place, not HTML but the syndication file – commonly called a *Web feed* or simply *feed*.

What for? The feed's file format (RSS or Atom) is more tightly structured than HTML and is thus machine-readable to a far greater degree. You can now have especially designed applications, called *feed readers*, that will subscribe to such a feed and check its location for updates every hour or so. If a feed has updated since the feed reader last checked, the reader grabs it and makes it available to you. In your feed reader, you can now gather content streams from all over the Web – from weblogs, newspapers and anyone who updates frequently and offers their content in this alternative format, and you can read it all in one place.

Popular feed readers include // Bloglines.com and the Google Reader – you might want to take a look at //Twitter Times as well, which is an aggregator that Melissa has just recommended to me.

The common thread here, the thing that ties in feed readers with the other things I've touched upon in this lecture is that, again, they provide an interface to the database, an interface that prioritises the stream of new stuff over the static hierarchy of persistent archives. With this new layer of technology you no longer code and upload, you automate and post. This change created new interfaces that re-defined today's experience of using the Web.

// That's it.

// Thank you.

// The tutorial, starting at half past three, will be held at Chadwick, as per usual. I've written up

your instructions on an old-fashioned, hand-coded XHTML page, which you will find at this address:

<http://tawawa.org/ucl/g0017.html>

Also on the screen is an e-mail address where you can reach me if you'd like for any feedback or discussion.

Off you go to Chadwick, see you over there!