

TRAINING XML USING SCENARIO-BASED INSTRUCTION

Barry Krusch

Imagine that your company just hired some people to do a project in XML, with just one slight problem: when you asked your new recruits if they knew XML, they said “huh?”. Further questions revealed they didn’t even know the meaning of *HTML*, a much simpler language. Oops.

The deadline is approaching, so you have to act fast. Better get a book on XML, and quick, to give to your not-so-techies to get them up to speed.

You go to a bookstore, and purchase one of the standard books on the subject. You take the book back to your office, and flip to a page with this paragraph:

This object represents a parsed or parsed entity as declared with an <!ENTITY . . .> element in the DTD. However, it does not provide a reference to the entity declaration. The W3C DOM recommendation does not define an object in version 1.0 that models the declaration of entities. In the following tables, ‘Ext’ indicates properties and methods that are extensions to the W3C Object model.

Uh-oh. You have just run into one of the main problems with technology training, which is that authors on XML will typically know all about “W3C” and “DOM recommendations” and other XMLspeak, but know nothing about how to *teach* what they know. They talk in their language, which you don’t know, and they forget that their primary mission is to first *teach* you their language. So you’re left in the dark.

Well, reading that isn’t going to save anyone time. But maybe there’s other material in the book which is more accessible. You turn to another page and you come across this tidbit:

If an XML document declares both inline (the internal subset) and an external DTD (the external subset), the declarations within the internal subset of the DTD take preference over any external ones where they refer to the same element type. There are two reasons for this, both dictated by the XML 1.0 specification:

The parser works with the first definition it is given for an element type; and

In the case where the document has both internal and external DTD subsets, although the reference to the external subset is given before the square brackets containing the internal subset, the parser doesn’t insert the contents of the external subset till after those of the internal subset.

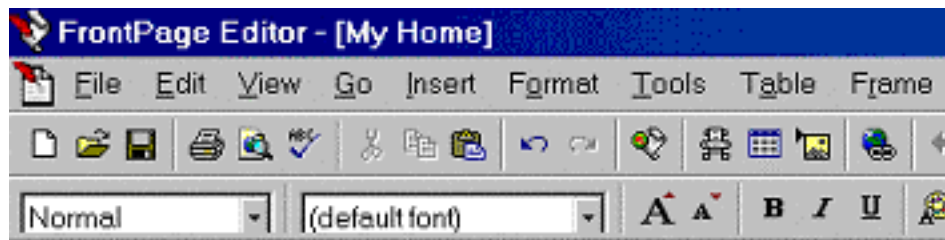
ZZZZZZZ . . . That’s the sound of one mind snoring.

Frustrated, you decide to browse the Web for an XML primer, and lo and behold!, the first site you come across seems to be just what you’re looking for. It’s even aimed at

people who know nothing about HTML, so it's bound to be easy enough for your audience. The site uses images with voiceovers. Here's how it goes:

XML TRAINING: THE EASY WAY

You're getting ready to sell your house, and you want to advertise it on the Web. You need to create a web page, so you buy a Web-page design program and start writing:



**This modern townhouse located in San Francisco
has five bedrooms, and a beautiful view.**

Not a bad start, but a bit boring. You want to make your page more interesting. You think “five bedrooms” and “San Francisco” are important, and you want them to stand out. You want to make them darker, which is universally known as “bold”, and you do that by first selecting the words you want to make bold, and then clicking the “B” on the toolbar:

[ANIMATION]

That was pretty easy. But can more be done? You notice another tab running along the bottom of the screen which says “HTML”:



You decide to click on it and see what's behind the scenes.

HTML

After clicking the tab, here is the most important part of what you see:

```
This modern townhouse located in <b>San Francisco</b>  
has <b>five</b> bedrooms, and a beautiful view.
```

Aha, now you see how the browser knows how to make the words bold. It surrounds the words with “tags” derived from a computer language called HTML. HTML stands for *Hypertext Markup Language*. It's *hypertext* because it lets you create links, words that if clicked, will take you to relevant Web pages, and it's a *markup* language because you're *marking up* your document in a language that your browser can understand.

You see, computers are much less intelligent than humans, but even humans need instructions about what you want to do. If you want a word on your web page to appear

dark, you have to have a language that tells your browser "make this word dark". In HTML, this language consists of *tags*, formatting instructions to the computer which are distinct from your content.

And how does the computer know how to separate mere formatting instructions from your precious prose? Because these instructions — *tags* — are enclosed in *angle brackets*, the first pointing left (<) and the second pointing right (>), with some type of code in-between, like this:

That type of tag, *left bracket* then *code* then *right bracket*, is called a *start tag*.

There is also another type of tag called an *end tag* which is also enclosed within left and right angle brackets, but has an additional slash (/) after the first angle bracket, and its humble role in life is to indicate unambiguously "stop this type of formatting *now!*"; for example,

OK, you know what a tag is and what *start* and *end* tags are, so let's put together the big picture of how your browser interprets your bold tags.

[BELOW PARAGRAPH ILLUSTRATED WITH ANIMATION]

When your browser comes across the first angle bracket of your start tag, it thinks, "OK, looks like a tag is coming up." And when it comes across the second and final angle bracket of your start tag, it knows for sure that this entity is a *tag*. It goes back to look between the brackets, and extracts the tag code, which in this case is *b* (of course, the software engineers who wrote HTML could have chosen another letter or word to be between the angle brackets, such as *bold*, but they chose the letter *b* instead, probably because *b* is short, and so a lot easier to type).

The browser, realizing that *b* is a legitimate tag in HTML, does what it is supposed to do, and makes that type darker, and continues to do so until it comes across the bold end tag (the one with the slash, remember?). And when it does come across the bold end tag, it stops what it's doing, and returns the look of the type to its normal, albeit boring, self. So when your browser displays it to you, you see this:

**This modern townhouse located in San Francisco
has five bedrooms, and a beautiful view.**

And that's the result of just one tag, the diminutive but powerful *bold* tag. As you might well know, you can do much more than change the emphasis of words, yes indeed. You can change their size, their shape, their overall appearance, where they are located on the page, and on and on, limited only by your time and your taste (or your client's taste).

That, in a nutshell, is one markup language: HTML.

HTML PROBLEMS

Okay, you mark up the document with these and other HTML tags, and then post it on the Web, and now you wait for a buyer. And wait. And wait. Hey wait, no one is calling you. Why not? Didn't you make your page more interesting with your *bold* tags?

Sure. But there's one problem: no one can find you. What's wrong? Didn't you get five bedrooms and San Francisco to stand out? Doesn't that help someone who's looking for a five bedroom house in San Francisco?

No, it doesn't. Let's find out why.

You see, there are literally millions of web pages out there, so apart from sheer luck, the only likely way a buyer will find you is through a search engine, a computer program which makes an index of web pages, and then lets you search that index.

But search engines aren't smart: even if a search engine knows about your page, the search engine most likely won't send your page to the top of the results. If a potential buyer goes to the search engine AltaVista.com and types "five bedrooms" and "San Francisco, California", here are the results returned:

1. [Yahoo! Business and Economy>Companies>Travel>Lodging>Bed and Breakfasts>B](#)

Help - More Yahoos. Home > Business and Economy > Companies > Travel > Lodging > Bed and Breakfasts > B.

All sites. Asia sites only. This category...

URL: asia.yahoo.com/Business_and_Economy/Comp...Pages/wp_2.html

Last modified on: 7-Mar-2000 - 80K bytes - in English

[[Translate](#)] [Facts about: [Yahoo Inc](#)]

2. [Self-Catering Accommodations in Lanzarote and Fuerteventura](#)

Premier Holidays, Lanzarote and Fuerteventura - the best on-line selection of rental holiday and vacation accommodation in the Canary Islands. Find...

URL: www.premier-holidays.com/cgi-bin/Propert...erty_lister.asp

Last modified on: 22-Jan-2000 - 36K bytes - in English

[[Translate](#)]

3. [3.2.98 Fortune Traveler: Cool Places for Hot Times](#)

Where to Dine Zagat restaurant guide Air Travel Frequent flyer deals Services Weather Currency converter Language translator Stock/Fund Quotes....

URL: www.pathfinder.com/fortune/roadwarrior/1...02rwarrior.html

Last modified on: 18-Jan-2000 - 17K bytes - in English

[[Translate](#)] [Facts about: [Time Inc](#)]

Search: [IHRFF](#)
[Books](#) [Music](#) [Mi](#)
[Toys](#) [Electroni](#)

CDNOW
[Buy Music](#)
[Find Any Artist](#)
[Buy CD's](#)
30% Off Top 100

AUCTIONS
[amazon.com](#)
[Find Great Items](#)
[at Amazon Auction](#)
[BID NOW!](#)

Your house isn't there. Nor, in fact, is *any* five bedroom house for sale in San Francisco. That's because the search engine can't separate Web pages describing homes for sale from pages which just happen to have those words listed on them. But if it could make that distinction, a potential buyer's search might return something like this:

Your Real Estate Listings Search Results

Documents 1 to 10 of 288



159,900.00, Bd, Ba, 1436SqFt, 1506 NORTHWOOD BLVD, ROYAL OAK, ABSTRACT



159,900.00, 4Bd, 2Ba, 1436SqFt, 1506 NORTHWOOD BLVD, ROYAL OAK, MI



164,900.00, 4Bd, 2Ba, 1838SqFt, 1904 BONNIE VIEW, ROYAL OAK, MI



102,000.00, 3Bd, 1Ba, 1400SqFt, 484 CAMBOURNE, FERNDAL, MI



109,900.00, 3Bd, 2Ba, 1150SqFt, 912 BUTTERNUT, ROYAL OAK, MI



119,850.00, 3Bd, 1Ba, 1400SqFt, 4167 LINDOW, STERLING HEIGHT, MI



88,900.00, 3Bd, 2.5Ba, 1026SqFt, 18200 BUCKHANNON, ROSEVILLE, MI



94,900.00, 2Bd, 1Ba, 962SqFt, 28338 LORENZ, MADISON HEIGHTS, MI



132,900.00, 3Bd, 1Ba, 1206SqFt, 2934 ALTADENA, ROYAL OAK, MI



116,500.00, 3Bd, 1Ba, 1700SqFt, 16042 EDMORE, DETROIT, MI

Page 1 of 29

Now that's a smart search. The results are right on point. So how can you help the search engine figure out the *meaning* of the document — its purpose, its role in the Web universe, that you are advertising a house for sale, thus enabling a potential buyer to find your page?

Since the language you're writing in is HTML, you might think that the solution will come from the proper use of tags. You might think, "Okay, I should have used a tag called <bedrooms> and a tag called <city>, like this":

This modern townhouse located in <city> San Francisco </city> has <bedrooms>5</bedrooms> bedrooms, and a beautiful view.

Then, a search engine could find any document with the <bedrooms> and <city> tags, and any buyer looking for a five bedroom house in my city would zero right in on me".

This is an excellent idea, but after doing a little research you see that this just isn't possible: HTML doesn't let you write tags that say "this is how many bedrooms I have", or tags that say "this is how much I'm charging for my house", or tags that say "this is where my house is located."

HTML is a language written by software engineers who wanted to describe the *look* of Web pages, not the purpose to which people like yourself would put them. And when you think about it, who could have written, or could write, a language which ultimately would have to see into people's minds?

INTRODUCTION TO XML

What you need instead of a pre-defined language like HTML is an infinitely flexible markup language, a language that allows you to write your own tags.

And that brings us to XML, which stands for Extensible Markup Language, and which was invented to solve problems just like this. XML is extensible, because you can *extend* the language by writing your own tags.

SCRIPT	INSTRUCTION	IMAGE
For example, just like you put <code></code> in HTML,	Fly out bold tags.	This modern townhouse located in <code></code> San Francisco <code></code> has <code>5</code> bedrooms, and a beautiful view.
you can put <code><bedrooms></code> and <code><city></code> into your XML document.	Fly in XML tags.	This modern townhouse located in <code><city></code> San Francisco <code></city></code> has <code><bedrooms>5</bedrooms></code> > bedrooms, and a beautiful view.

This is a vast improvement. Note how the meaning of your terms is much clearer. "San Francisco" is a city, and the "5" here refers to the number of bedrooms.

This is a good start, but this is not yet a valid XML document. More lines need to be added. The first line to add is a line that tells your browser that this is a document in the XML language, as opposed to some other language like HTML.

To define this page as an XML document for your browser, you need to add the following line at the very top of the page, with no blank lines ahead of it:

```
<?xml version="1.0" ?>
```

```
This modern townhouse located in<city> San Francisco </city>has  
<bedrooms>5</bedrooms> bedrooms, and a beautiful view.
```

This line has the following characteristics:

SCRIPT	INSTRUCTION	IMAGE
<p>1) Lines which begin with “<?” and end with “?” are called processing instructions; therefore, this line is an instruction to the browser about how to process the document.</p> <p>2) The first word after the “<?” is the name of the instruction, which is “xml”. “xml” must follow immediately after the “?”, with no space between them, and must be in lowercase.</p> <p>3) All XML documents have to specify what version of XML the document will follow, and this is done by the word “version” which is followed by “=”, and then the version number within quotation marks. Here the version number is “1.0”.</p> <p>4) The line is ended with ?>.</p>	<p>Fly in the separate parts as the script progresses.</p>	<p><? xml version= “1.0” ?></p>

And here’s your XML document so far:

```
<?xml version="1.0" ?>
```

```
This modern townhouse located in<city> San Francisco </city>has  
<bedrooms>5</bedrooms> bedrooms, and a beautiful view.
```

The DTD

Again, a good start, but still not complete. You see, thinking of tags and then putting these tags in this XML document are just the first steps towards creating what is referred to as *valid* XML. To create valid XML, you need to tell the browser the overall structure of your document — that is to say, what the tags are, and how they relate to each other — and you also need to explain what type of information the tags are going to contain. This listing of the tags you’re going to use in your document — how they relate to each other, and the type of information they contain — is known as a DTD, which stands for

Document Type Definition

A DTD is included in your file directly at the beginning of your XML document, or can be linked to an outside source, or a combination of the two.

SCRIPT	INSTRUCTION	IMAGE
<p>In the following example, we will insert our DTD at the top of our XML document</p>	<p>MOVE UP</p>	<pre><?xml version="1.0" ?></pre>
<p>between these two lines:</p>	<p>MOVE DOWN</p>	<pre>This modern townhouse located in<city> San</pre>

		Francisco </city>has <bedrooms>5</bedrooms > bedrooms, and a beautiful view.
--	--	---------------------------------------------------------------------------------------

When we're done, here's what the DTD will look like:

```
<?xml version="1.0" ?>
<!DOCTYPE webhomeadvertisement
[
  <!ELEMENT city ANY>
  <!ELEMENT bedrooms ANY>
]
>
```

**This modern townhouse located in<city> San Francisco </city>has
<bedrooms>5</bedrooms> bedrooms, and a beautiful view.**

Now, don't let this frighten you. We are going to explain it carefully.

The first step in writing a DTD is to include the following writing:

```
<!DOCTYPE DTDname
>
```

Let's break this down.

SCRIPT	INSTRUCTION	IMAGE
We are instructing the browser that this is a DTD by beginning with the left angle bracket and exclamation point.	Fly in.	<!
Next, comes the word DOCTYPE. Remember, DTD stands for document type definition, and DOCTYPE is a short way of saying document type. This word DOCTYPE must follow immediately after the exclamation point, without a space between them.	Fly in.	<!DOCTYPE
We next tell the browser what type of document this Web page is going to be, which we're calling <i>webhomeadvertisement</i> , to remind us of the purpose of this page. A space goes between those.	Fly in.	<!DOCTYPE webhomeadvertisement
Finally, we'll type some extra space between the brackets to make room for what is going to follow,	Fly in.	<!DOCTYPE webhomeadvertisement >
and a right angle bracket to close things out. That leaves us with this:	Fly in.	<!DOCTYPE webhomeadvertisement >

Next, we have to add two brackets, since definitions of XML tags go between brackets. We will indent the brackets to make them easier to read, and we'll put some extra space between them to leave room for our terms:

```
<!DOCTYPE webhomeadvertisement
 [
 ]
 >
```

Now we are going to add our first term. XML definitions start with <! and end with >. After the first <!, you add the word ELEMENT, completely capitalized. You then follow with the name of your XML term, in this case "city", and then a keyword which indicates what type of information the tags going to contain. The most flexible keyword is ANY, and that's what we will use here. You then close the definition with the >:

FLY IN THE FOLLOWING ELEMENTS

```
<!DOCTYPE webhomeadvertisement
 [
   <!ELEMENT city ANY>
 ]
 >
```

We will do the same thing for our second definition. Again, after the first <!, you add the word ELEMENT, completely capitalized. You then follow with the name of your second

XML term, in this case “bedrooms”, and then the keyword which indicates what type of information the tags going to contain. Again we close our definition for our “bedrooms” tag with the >:

```
<!DOCTYPE webhomeadvertisement
[
  <!ELEMENT city ANY>
  <!ELEMENT bedrooms ANY>
]
>
```

We are almost complete. We need to add another element, which is called the *root element*. The start tag for a root element goes immediately after your DTD, and the end tag is the last line of your document. The root element must have the same name as your document type, is since our document type is called **webhomeadvertisement**, we'll call it that.

```
<!DOCTYPE webhomeadvertisement
[
  <!ELEMENT webhomeadvertisement ANY>
  <!ELEMENT city ANY>
  <!ELEMENT bedrooms ANY>
]
>
```

And now we will insert our root tags:

```
<?xml version="1.0" ?>

<!DOCTYPE webhomeadvertisement
[
  <!ELEMENT webhomeadvertisement ANY>
  <!ELEMENT city ANY>
  <!ELEMENT bedrooms ANY>
]
>
```

<webhomeadvertisement>

This modern townhouse located in<city> San Francisco </city>has
<bedrooms>5</bedrooms> bedrooms, and a beautiful view.

</webhomeadvertisement>

And here's our completed document, valid XML:

```
<?xml version="1.0" ?>

<!DOCTYPE webhomeadvertisement
[
  <!ELEMENT webhomeadvertisement ANY>
```

```
<!ELEMENT city ANY>
<!ELEMENT bedrooms ANY>
]
>
```

```
<webhomeadvertisement>
```

```
This modern townhouse located in<city> San Francisco </city>has
<bedrooms>5</bedrooms> bedrooms, and a beautiful view.
```

```
</webhomeadvertisement>
```

Coda

Now, we have been working with very simple examples, but there's no limit to the tags you can invent. If you want to tell future buyers how many square feet your home has, invent a `<squarefootage>` tag. If you want to tell future buyers how many bedrooms you have, invent a `<bedrooms>` tag. All that's required is thinking about anything anyone would possibly want to know about a house, and then inventing tags for it.

Of course, we have been working with a very simplified version of XML for educational purposes, and the above XML document should be rewritten to allow for greater precision and structuring. For example, the ANY keyword should be replaced with a different XML keyword. But we will tackle that in a different module.