



Capstone Courseware, LLC

33 Boylston Street
Jamaica Plain, MA 02130

877-227-2477
capstonecourseware.com

XML Schema

**Will Provost
Robert E. Marcum**

Instructor's Guide

Revision 1.0.6



Revision Notes

Revision 1.0.6 updates the coursebook to current Capstone standards, and enhances the lab software with a few additional exercises, focusing on practice in basic content modeling in the second and third chapters.

- There is now a detailed table of contents.
- A new Lab 2C on unions and enumerated types.
- A new Lab 3B on various content-modeling techniques.
- A new challenge-lab 3D to build a moderately complex schema from scratch.
- The Cars VIN RE bug mentioned in 1.0.5 below had not been completely stamped out. Now, it has been.

Revision 1.0.5 implements the following fixes in the XML Schema modules:

- The Cars schema had a bad regular expression for the VIN type.
- The Venn diagram showing “types of types” has been clarified with the addition of dots showing possible instances in the type space. The previous diagram might have been misleading, because there were several subsets shown which could not have any members – for instance there are no built-in types which are neither primitive nor derived.
- News schema had an **xs:all** with a multiply-occurring particle.
- News schema had a bad regular expression for the Filename type.
- Movies/Namespaces had bad key references “Catch22” which were caught only by the named-NS schema. Now that these have been fixed, problems with the XPath in the default-NS schema are invisible, but the book mentions them at the end of the demo.
- Movies/Namespaces had a lingering Showtimes list with bad data.
- Miscellaneous typos and formatting problems.

Revision 1.0.4 includes fixes for some time and date values in instance documents.

Revision 1.0 is the initial public release.





Course Overview and Philosophy

This course focuses on XML document design, or how to define XML vocabularies. While the course does not assume any programming or database knowledge on the student's part, the nature of the subject will quickly drive the discussion towards topics like object and type modeling, OO analysis and design, relational data design, and others most familiar to server-side developers. It is expected that students interested in this course will be preparing for document-centric tasks such as publishing systems design or management of XML documents as externalizations or archives of relational databases. Those who need basic familiarity with schema use in authoring XML should get what they need from Course 501, "Introduction to XML," or possibly study the first few chapters of this course for a bit more depth.

All of Capstone Courseware' Core XML modules present what might be called "pure XML", by which we mean two things. Firstly, everything in these modules is based strictly on W3C specifications, without any vendor-specific extensions. Secondly, no knowledge of any particular programming language or other external technology is required to participate fully in the training. For this particular course, it is likely that students will have some database background, but, again, this is not required. As with almost any XML training, the instructor will have to be prepared for a diverse student body. We have tried to make as few assumptions as necessary in writing each module, keeping terminology simple and introducing parallels to other architectures and paradigms only as asides in the text.

More specific discussions of module and chapter contents, and suggestions for effective teaching, appear on the pages following the timeline below.





Timeline

Day 1

XML Schema, Chapter 1
XML Schema, Chapter 2
XML Schema, Chapter 3

Getting Started with XML Schema
Simple Types
Complex Types

Day 2

Advanced Schema, Chapter 1
Advanced Schema, Chapter 2
Advanced Schema, Chapter 3
Advanced Schema, Chapter 4

Keys and Key References
Complex-Type Extension
Namespaces and Schema
Using Schema in XML Applications





The XML Validator

As you will see in the coursebook and the course setup guide, Capstone Courseware provides a standard tool to support XML parsing and validation in the classroom. This is a simple wrapper around the Xerces parser from Apache: a Java application implements a command-line interface; **parse** and **validate** scripts further simplify usage; and the pluggable tool configuration integrates this with the Crimson text editor. For most classes you should find this basic setup is most appropriate, but here are a few additional notes on the tools should you want to customize a particular class. Note that Capstone Courseware cannot provide technical support for any of these options; if you are at all uncertain you should stick with the standard setup.

- If you don't prefer to use the Crimson editor, of course you won't need the tool pack, either. Most serious text editors (anything much better than Notepad!) have some ability to trigger external scripts from their Tools menus, and you should find that the **parse** and **validate** scripts work easily from TextPad, UltraEdit, Syn, etc. – or for that matter from more sophisticated IDEs such as Visual Studio or Eclipse, if you don't like the XML tools built into them.
- The standard setup assumes only a Java runtime, and hence includes the Xerces implementation as part of the download. There is an alternate version (also mentioned in the online setup sheet for this tool) that can be used when Xerces is already present thanks to a previous J2EE, JWSDP, or JAXP installation. If you are comfortable with Java class path configuration, you may also choose to modify the **parse** and **validate** scripts to include the Xerces JAR from any other location on a student machine.
- Scripts **parse_pause** and **validate_pause** are also available. These are handy for attaching to the XML file type in Windows as "actions." So you might assign each of these scripts so that they appear as right-button menu items for an XML document in Windows Explorer. These two scripts hold the resulting DOS console on the screen until the user presses a key – just for this purpose.



Module 1 – XML Schema

The XML Schema recommendation is large and dense, and it's difficult to get a true working knowledge of it in a one-day module. This first of two schema modules should take the student to the point at which he or she can develop simple vocabularies (and, especially, port existing DTDs to schema), and can read schema documents reasonably well.

Chapter 1 – Getting Started with XML Schema

Introduces the Schema recommendation and its relationship to XML, DTDs, and namespaces, and gets students started with some simple hands-on exercises in defining schema and associating them with instance documents. We've discovered that it is impossible to understand either simple or complex types to any meaningful depth without learning something about both, so in this chapter we develop just a minimal knowledge of complex types, firstly to allow students to pursue a sort of "hello, schema"-level exercise and secondly to set up the progression over the next two chapters through detailed study of simple types and then complex types.

Chapter 2 – Simple Types

Simple types are covered in detail. By the end the student should be comfortable with the structure of the typing system, with the distinctions between built-in, primitive, derived, and atomic types, and with the basic process of simple-type restriction to create new types. We do not provide details on every built-in type; students can find everything they need to know about value spaces, fundamental and constraining facets, and lexical representations in the recommendation itself (part 2).

Note that the standard validating tool we've chosen, XSV, is a little weak on simple-type validation, and so you will have to reinforce some concepts on which the tool will not support hands-on work. Specifically, constraints on pattern, numbers of digits, and date/time representations are not enforced. The tool is much stronger on content-model validation, and so this chapter will be the weak spot.

Chapter 3 – Complex Types

Now we turn to look at complex types, and the beginnings of content-model building. This chapter mostly covers the means by which a schema author can state restrictions on element grammar similar to those that can be specified in a DTD: element and attribute definitions, model groups and particles, occurrence constraints, and model and attribute group definitions as a near-replacement for parameter entities. (Experienced DTD authors may be bothered by the lack of entities in schema; point out that this general text-replacement capability lacks structure and is highly error-prone, and that the need for parameter entities is greatly reduced since schema provide so many good means of group and type reuse.)

Module 2 – Advanced XML Schema

This second module completes coverage of XML schema, with discussions of identity constraints, type extension, and namespaces, and a brief chapter on design issues.





Chapter 1 – Keys and Key References

XML schema are of course capable of much more precise models than DTDs, and nowhere is the gap greater than in managing element relationships. The ID/IDREF mechanism, while still available for compatibility, is outdone in a number of ways by schema identity constraints. The first half of this chapter, though, is devoted to developing UML as a means of modeling XML data. UML may or may not be the language of choice here, but it serves this module not only as a notation system but in underscoring the usefulness of object-modeling concepts in XML schema design. Keys and key references, then, are introduced as a way to map an OO concept that the first schema module did not address, which is class associations.

If students seem ready for the analogy, another way to go at this chapter is from the perspective of relational database design. This mapping is studied more directly in Chapter 4, but here it can motivate and organize the discussion – again, if students have enough grasp of RDB concepts to start with. Otherwise it will be a distraction.

Chapter 2 – Complex-Type Extension

We begin by returning to the UML-for-XML discussion, to introduce another OO concept that has thus far been absent from the schema-design discussion: specialization. DTDs, of course, are short on mechanisms for type reuse, but schema have many means, some of which were studied in the first schema module. This chapter focuses on complex-type derivation, especially extension. Extension of simple types and restriction of complex types are also covered briefly, but the extension mechanism is emphasized as the most likely means of general type reuse in complex schema designs for enterprise software. Various features of the XML schema type model are then (conveniently) mapped from corresponding OO concepts and UML notation: abstract types, complete specialization via the final attribute, and use of the block attribute to eliminate polymorphism. Throughout this chapter, and the module, really, it's important to keep a theme going, which is using the XML schema components and techniques to model data according to some existing traditions – object-oriented design and relational data design.

Chapter 3 – Namespaces and Schema

It's finally time to tame the namespace beast. Through the previous modules, namespaces have been treated lightly because the topics at hand could be understood without them. Now, however, as we've developed in the student the ability to design sophisticated document models for complex XML applications, rigorous use of namespaces must be understood, and should be encouraged.





Probably the trickiest of a few possible confusions will be that between use of a namespace and the importing of a schema's type information for that namespace. Instance documents can only achieve the former, but in schema documents it is possible to reference a namespace without gaining access to the schema information for that namespace: that is, to reference the namespace without importing the schema. When schema includes and imports are reached, highlight them as the only ways of, for instance, extending a type in another schema. Students otherwise may latch on to the idea of referencing a namespace as being enough to empower a schema document to expand on that space's type information. Also, this is a good time to point out that many XML tools do not yet handle namespaces well, and that other W3C specifications have yet to completely embrace XML schema in general. (For instance XPath does not support all the schema types just yet.)

Chapter 4 – Using Schema in XML Applications

This is a sort of best-practices chapter for document design. Feedback on the content here would be especially welcome. Depending on student background, this material may be anywhere from uninteresting to revelatory.

What will probably be of interest and use to everyone is the latter section on using XSLT for document validation. This can be part of a more general wrap-up discussion of how XML schema can best be used in application architectures. Schema in particular (among XML technologies) can be seen as useful to software development (i.e. part of a build process) as well as to a running application (for instance, validating client input at runtime). In both cases, the goal should be to apply as strong a set of validation rules as is reasonable for the document class in question, so as to catch errors early and clearly. Schema can be seen as the first, best approach to this, but there are certainly rules that it cannot express, and XSLT turns out to make a great second layer. Last comes code-level validation, and it's hard to say too much of a specific nature about this, but it's worth discussing from a high level as another layer in the application's validation architecture.





Troubleshooting and Tool Tips

First, be sure to be familiar with the lab and environment setup instructions in the Table of Contents. Assure that the instructions in the course Setup Guide have been followed, and that you know the locations of the installed tools. If, having followed those steps, a classroom machine is failing to run demos or lab answers correctly, here is a list of items to consider and to doublecheck:

- Make sure the **c:\Capstone\XMLTools** directory is on the path to use the **parse** and **validate** scripts from a command console.
- If you get errors that certain classes are not found when you run either of these scripts, be sure that you have either downloaded and set up the standard XML Validator (which includes Xerces) or have Xerces set up on your system and have modified the scripts to include the Xerces JAR in your class path. See the earlier section on using the XML Validator.
- Running **validate** on a file with no DTD or schema reference will generate errors; use **parse** instead.
- There is a little kludge in how the tool validates schema documents themselves: it checks the file for the extension **.xsd**. If this is found, the tool goes to the extra trouble of loading the “schema for schema” into the parser before validating. This is unnecessary for ordinary documents. In the normal run of this course, this is not known to be a problem, but it’s certainly possible for the tool to trip up if you point it at a non-schema doc with the **.xsd** extension or a schema doc without it. There are no plans to clean this up – the tool is only meant to support Capstone XML courses, after all, and was never offered as an industrial-strength parsing tool.





Feedback

We truly do welcome feedback, both of a specific nature (pointing out mistakes) and general suggestions. For the former sending email with a numbered list of corrections would be most helpful.

Please send feedback to:

Will Provost
Capstone Courseware
<mailto:provost@capstonecourseware.com>
www.capstonecourseware.com

