

OREILLY'

XML Publishing with Adobe InDesign

By Dorothy J. Hoskins

Copyright ©2010

ISBN: 978-1-449-39857-6

Released: October 1, 2010

Covers InDesign CS3-CS5

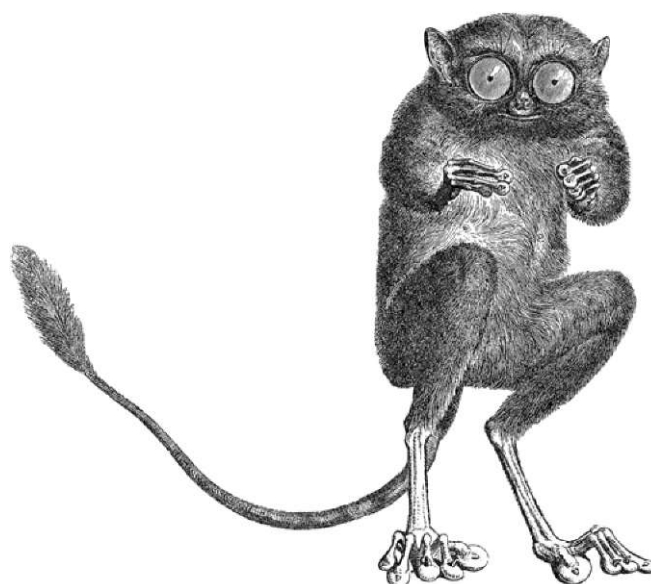
From Adobe InDesign CS2 to InDesign CS5, the ability to work with XML content has been built into every version of InDesign. Some of the useful applications are importing database content into InDesign to create catalog pages, exporting XML that will be useful for subsequent publishing processes, and building chunks of content that can be reused in multiple publications.

In this Short Cut, we'll play with the contents of a college course catalog and see how we can use XML for course descriptions, tables, and other content. Underlying principles of XML structure, DTDs, and the InDesign namespace will help you develop your own XML processes. We'll touch briefly on using InDesign to "skin" XML content, exporting as XHTML, InCopy, and the IDML package. The Advanced Topics section gives tips on using XSLT to manipulate XML in conjunction with InDesign.

Shortcuts

Contents

Extended Contents.....	3
About This Book and InDesign CS.....	5
A Brief Foray into Structured Content (a.k.a. XML).....	6
InDesign XML Publishing: College Catalog Case Study.....	9
Importing XML.....	17
Tagging XML in InDesign.....	49
Looking Forward: InDesign as an XML "Skin".....	51
Exporting XML.....	53
Validating XML in InDesign.....	61
What InDesign Cannot Do (or Do Well) with XML.....	71
Advanced Topics: Transforming XML with XSL.....	74
ABrief Note about InCopy and XML ..	104
A Brief Note about IDML and ICML..	106
Summary.....	109
Resources.....	111



XML Publishing with InDesign CS

In this book, we will be referring to InDesign CS5, and previous versions of the program back to CS3, generically as "InDesign CS". When there are important differences in the version's XML features, we will indicate for which version the screen shot or other information applies. Many features are not different from one version to another. Generally the screen shots will be taken from InDesign CS5. We'll start with the assumption that you already know quite a bit about InDesign typographic styles and layout features because you want to use InDesign to do something with XML. In particular, you will need to understand the role that paragraph and character styles play in making InDesign documents that are consistently the same in their typographical presentation throughout a given document, or a set of documents using the same InDesign template. (If you are new to these concepts, please refer to Adobe's InDesign CS built-in **Help>Styles** or see O'Reilly's **Adobe InDesign CS5 One-on-One**.)

The power that XML brings to the InDesign world is summed up in the word "interoperability," which means that the same content in XML format can be used in multiple applications or processes, and not solely inside InDesign. XML is typically used for creating HTML for web sites, but it can also be used to create rich text, PDF, or plain text files. XML does not inherently have "presentation styles"—how the content of an XML file looks depends upon the way that it is formatted and used by applications. The main purpose of XML is to provide a reliable structure of content so that it can be processed consistently once the application has rules for presenting the structure visually. (For more information on XML, see O'Reilly's *XML in a Nutshell, Third Edition*.)

For example, in a course catalog, we may have information that resides in a database in a set of tables (course descriptions, programs of study, faculty and staff directory, etc.). The information in the tables is the "content"; the way that it is organized in table columns, rows, and cells is its "structure." If we save the data as XML, it becomes the structured content that we need, but now it is no longer bound to the database application. It's ready to use and reuse in other applications, including InDesign CS5.

Note: InDesign has features for importing and working with data in comma-separated-values (.csv) or tab-delimited (.txt) text format. But XML provides for a much more complex information structure to be imported into InDesign, which we will explore throughout this Short Cut.

We'll look at how and why you might want to tag content as XML in InDesign and export it to use in other applications. A theoretical workflow for XML with XSLT

to create web page output will give you ideas for what you might want to do with your own InDesign documents.

Also, we will look at the methods you can use to get InDesign to automatically provide the right paragraph styles when importing XML. Besides InDesign's "Map Styles to Tags" and "Map Tags to Styles" dialogs, you can go further with the use of XSLT and the "namespaced" XML that is part of InDesign under the hood.

Extended Contents

Just because it's sometimes nice to find topics of interest faster:

XML Publishing with Adobe InDesign....1	Linking to external XML files..... 35
XML Publishing with InDesign CS..... 2	Creating text flows for the imported XML..... 36
Extended Contents.....3	The importance of "document order" for imported XML.....36
About This Book and InDesign CS.....5	Understanding InDesign's XML Import Options.....39
A Brief Foray into Structured Content (a.k.a. XML).....6	Using "Clone repeating text elements"40
InDesign XML Publishing: College Catalog Case Study.....9	Importing only elements that match structure.....41
Data-like Content Example: The Course Description XML.....11	Avoiding overwriting text labels in the placeholder elements.....43
Topical Content: The Handbook XML ..13	Deleting nonmatching structure, text, and layout components.....45
Iteration and refinement.....15	Importing Images.....47
Net results: vast improvements in understanding and speed.....15	Inline image imports.....48
Importing XML.....17	Tagging XML in InDesign.....49
Doing It Adobe's Way: The Placeholder Approach.....17	The Case for Tagging Content: Why You Need XML.....49
Model the XML you want.....18	Tagging for Import.....50
Importing XML into placeholders.....22	Tagging for Iterative XML Development50
An aside: the scary Map Styles to Tags dialog message.....29	Working without an initial DTD..... 50
Mingling non-XML and XML content in a text flow.....31	Looking Forward: InDesign as an XML "Skin".....51
Exporting XHTML when XML is in your InDesign file.....33	Exporting XML.....53
Doing It Your Way: Using the Options for Your Own Process.....35	Marking Up (Tagging) Existing Content for XML Export.....53
Import XML using only Merge, no other import settings.....35	The Special Case of InDesign Tables (Namespaced XML).....53

Examining the table.....	54	XSLT for Wrangling XML; XML Scripting for Automating XML Publishing	74
Tagging Images as XML in InDesign	59	XSL: Extracting Elements from a Source XML File for a New Use.....	76
Image Options in the Export XML Dialog	59	XSL: Getting the Elements to Sort Themselves.....	77
Validating XML in InDesign.....	61	XSL: Getting Rid of Elements You Don't Want.....	79
Why validate?.....	61	Creating Wrappers for Repeating Chunks	81
How to Validate XML in InDesign	61	Making a Table from Element Structures	85
Loading a DTD and getting the correct root element.....	62	Upcasting Versus Downcasting.....	88
Authoring with a DTD.....	62	Upcasting from HTML to XML for InDesign Import.....	92
Dealing with validation problems.....	63	Downcasting to HTML.....	93
Occurrence and sequences of elements	66	Generate a Link with XSLT (Not Automated).....	99
Validating outside of InDesign	67	Adding Useful Attributes to XML.	100
Duplicating structure to build XML....	68	A general formula for adding attributes	101
Cleaning up imported XML content ...	68	Generating an id attribute for a div ...	101
Fast and Light Credo: Develop Now, Validate Later.....	69	Use of the lang attribute for translations	102
Iterating the information structure and DTD.....	69	Creating an image href attribute	102
What InDesign Cannot Do (or Do Well) with XML.....	71	A Word about Using Find/Change for XML Markup in InDesign.....	103
The 1:1 Import Conundrum.....	71	ABrief Note about InCopy and XML...104	
Bad Characters.....	71	A Brief Note about IDML and ICML..106	
Inscrutable Errors, Messages, and Crashes	72	Summary.....109	
The Devilish DTD suggestions.....	72	Resources.....11	
Exporting from the element with the included DTD will not be valid.....	72	InDesign Resources.....	1
Don't make InDesign "think" too hard on import or export with XSL.....	72	XML Resources.....	1
InDesign CS5: XML Structure option for exporting XHTML.....	73	XSLT Resources.....	1
Advanced Topics: Transforming XML with XSL.....	74	Acknowledgements.....	1
An Aside Regarding Scripting InDesign and XML Rules-based Publishing.....	74		

About This Book and InDesign CS

While I had the opportunity to work with both InDesign CS2 and CS3 and XML, the release of CS3 in May 2007 occurred almost simultaneously with the first publication of this Short Cut. Consequently, I wrote the first version of the book based on CS3 and CS2. In this new version, I am updating the information and screen shots as needed to bring the book up to InDesign CS5.

Chief among the new features introduced in CS3 and retained in CS4 and CS5, is the ability to apply XSL transforms to XML when importing or exporting from InDesign. I have included some XSLT examples in the Advanced Topics section, but there is much more to explore, such as the ability to automate XML processes using scripts. Scripting requires advanced understanding of both XML structures and programming, so what I cover here will just provide a taste of the possibilities.

I assume that InDesign will perform virtually the same on Mac OS as on Windows, as Adobe makes InDesign cross-platform compatible. However, you should be aware that only Windows was used for the development of the test materials for this publication. If you use InDesign on a Mac or in a mixed-OS environment, there is the possibility that something might not work as described in this **Short Cut**

Adobe provides for forward migration—opening CS in later versions than when they were created—which appears to have no negative impact regarding XML processing. However, the table model for XML was completely redesigned in CS3, and table XML may not be usable in early versions of InDesign CS without some XSL transformation.

Adobe also provides for backward compatibility to some extent. You can save a CS3 file as an InDesign Interchange file (.inx) that can be opened in CS2. You can save CS5 and CS4 files in IDML format, and most CS5 features will be available when you open the file in CS4. Refer to the Adobe InDesign documentation for assistance with InDesign backwards-compatibility features and processes, especially new documentation for CS5.

My intent is to help InDesign users understand how to work with XML more than to help XML users understand how to work with InDesign. Thus, I include explanations of XML that may be unnecessary for those experienced with it. I hope that XML novices will be able to follow the examples, and XML experts will get ideas for venturing beyond the examples on their own.

A Brief Foray into Structured Content (a.k.a. XML)

Whenever we talk about extensible Markup Language (XML), we are talking about some kind of "structured content." In case you haven't been exposed to these concepts, we take a brief look at them before we dive further into XML and InDesign.

The first concept is that of structure, sometimes called "hierarchy." This is the organization of pieces of information into a grouping that makes sense to humans. For example, if you are going to describe a course within a college course catalog, at minimum you would give the course name and a brief description. To relate this course to the larger picture of getting a degree, you would want to provide information about the major that the course is part of, tell the prospective student how many credit hours the course counts for, and provide information about the prerequisites, if there are any.

Looked at from the top down, a college offers programs of study consisting of courses in a sequence. Course credits have to add up to the required number for the degree program.

If you draw the relationships as boxes that contain information, you might see that a program of study contains a set of repeating information blocks consisting of blocks of course names and descriptions, like this:

Program of study

(times N number of programs)

Description of program

Sequence of courses (credits)

- course 101, required (3)
- course 102, required (3)
- course 202, required (3)
- course 202, required (3)
- course 203, minor
- course 301, required (3) *etc.*

- elective A, elective B, elective C, *etc.*

Total credits required

Cumulative grade points acquired

Course *(times N number of courses)*

Name of course / course ID

Description of course

Notes (prerequisites, etc.)

A diagram of a possible course catalog structure

The second concept is "semantics," which is applying names to things so that they are meaningful to you and others. So rather than Titlemain, Titlesub, and List, you would use names that relate to the type of information you are organizing:

ProgramName, ProgramDescription, ProgramRequirements, CourseName, CourseDescription, Credits, etc.

Hierarchy and semantics are combined in structured content, and can be translated into an abstract model of XML elements, such as:

ProgramsOfStudy

```

└─ ProgramOfStudy
  └─ ProgramName
  └─ ProgramDescription
  └─ CourseSequence
    └─ CourseDescriptions
      └─ CourseDescription_Major
        └─ CourseDescription_Name
        └─ CourseCreditsHrs
        └─ CourseDescription_Text
        └─ CourseDescription_Fotnote
      └─ CourseDescription_Minor
        └─ CourseDescription_Name
        └─ CourseCreditsHrs
        └─ CourseDescription_Text
        └─ CourseDescription_Fotnote
  └─ ProgramRequirements
  └─ TotalProgramCredits
  └─ CumulativeGradePointAverage
  
```

Each piece of information that we want to identify and work with is given an element name. The top-level element (root) in the example at left is named <Programs of Study>, and consists of many individual <ProgramOfStudy> elements.

Repeating element blocks make up a <CourseSequence> element.

The names of elements can be very wordy to ensure that humans can read and understand what they mean, or they can be tersely named like Prg, Crs, and TCrd if mostly computer programs use them. XML element naming is dependent on who has to work with the XML and how.

Here are some general naming rules: element names can't start with a number, can't contain spaces, and can't contain certain "illegal" characters such as ?, >, &, and /.

A tree diagram of possible course catalog structure

If a structure of meaningful components will be used by more than one person or organization, it can be formalized with a set of rules, such as:

Every program of study must consist of a sequence of more than one of each of required major courses, required minor courses, and elective courses. Additionally, the course credit hours must add up to the total credit hours required to complete the program of study, and the grades received must cumulatively add up to the minimum grade average for the student to graduate.

A set of rules for the structured content is called a schema or a Document Type Definition (DTD). The rules can be simple or complex, depending upon the number of elements, and how they can be used (whether required or optional, how many times the element can occur, and within what contexts, etc.).

Rather than spend a lot of time exploring XML and DTDs at this point, we will consider them as part of the problem-solving process for creating a content creation and publishing workflow. There are many resources for learning about XML and DTDs online.

The key points to keep in mind are what you call the pieces of content (the element names), and how they are organized (the structure). These points are factors in setting up your InDesign import and export processes. The names of your elements can be the same as, or different from, the names of paragraph styles that you use in InDesign.

XML element attributes provide additional information, typically to enable finer distinctions among content that is basically the same. For example, in a staff directory, an attribute might be used to indicate a department head, so that when the person's name is shown, their name gets special typographical treatment in InDesign.

Unless you are using a DTD or schema developed by someone else, you can name elements and attributes in ways that are meaningful for your organization. That's why XML is "extensible"—you are not limited to just a defined set of elements like you would be with HTML for web pages.

If you are using a DTD or schema provided by another organization, you will have to learn how the elements and attributes in it create the kind of structure that you will work with in InDesign. We'll examine elements and attributes and their naming more in subsequent chapters.

InDesign XML Publishing: College Catalog Case Study

Most people look at InDesign as a layout tool for highly styled graphic designs that are rich with color and typographic controls. In some cases, users also have explored how to import data into tables or how to export InDesign as HTML. InDesign CS is fully capable of all these things, but if a person is exploring XML, it is usually because someone has said, "Hey, we need to use XML so that we can make web pages and PDFs and everything out of the same content." Perhaps the organization is already using XML for the web site, and someone has seen that InDesign can work with XML. Or someone has used InDesign and wonders how to extract the content from InDesign in a way that a web service or other application can use it.

In any event, although InDesign can do some pretty useful XML importing and exporting, Adobe does not see this as a feature intended for typical users. Their demos are business card templates and cookbooks, and making XML that will match what another application or process uses is not the focus of their examples. However, Adobe has provided a number of features in InDesign for importing, creating, and exporting XML, which we will cover in this Short Cut.

To get the most of the XML capabilities of InDesign, you should think about the bigger issues of the processes you have in place, the workflow that will help with it, and whether you need to create XML from content you already have in InDesign (export XML), or need to create InDesign documents from XML (import XML), or do both of these processes (bidirectional XML import/export).

Here, I will use as an example an actual project that needed import and export. The college course catalog consists of a number of chapters, such as:

- General information about the college, its history, and its program emphasis, as well as its academic calendar
- Financial aid, admissions criteria, application process
- Programs of study
- Course descriptions
- Student services, regulations handbook, policies and procedures
- Faculty and staff listing, directory and campus maps

Of these, some financial aid data, the course descriptions, and the programs of study were all stored in database tables. The content of the database was published directly to the college web site as HTML pages using ASP. The rest of the content was created by staff members who sent Word documents to the InDesign layout

person, and they did not exist in the database as text entries. The InDesign files were used primarily for the paper printed output, a bound catalog.

The goal was to make the database the "single source," with the web site and the printed catalog being two outputs of the same content. To get the current processes synchronized, content in InDesign would be added to the database, and content from the database would be passed into InDesign.

We were dealing with two different types of content in the catalog: some could be assigned neatly to table rows and cells in a database, and some was more narrative or organized in topics. Each of these types of content needed its own analysis and design process to achieve the XML import/export. Key issues and proposed solutions were these:

- Database content was extracted as plain text (separated into paragraphs), and given to the layout person in one large .txt file. The layout person imported the plain text, and then had to mark up every paragraph with the correct InDesign paragraph style. Because about two-thirds of the catalog content was in the database, this meant the layout person was manually marking up more than 130 pages of the catalog. The proposed solution was to provide database content to the layout person such that it would format itself automatically upon import into InDesign.
- On the other hand, all of the text about admissions, policies, registration, regulations and personnel was being created in Word docs. These docs were used as source material for the InDesign catalog by importing the Word docs. The text then was edited in InDesign, and finally was added to the database and web site via cut and paste operations from RTF files saved from InDesign. There were problems with getting changes on time and mistakes in editing that let to differences in the text outputs. The proposed solution was to provide the output to the database and web site developers such that it could be imported as rich text "blobs," but still have some semantic meaning that would assist in locating and reusing it. After the initial import into the database, the database programmer would provide a web-based form for editing, so that the database would be the ongoing "single source" for this content.

Both of these processes involved InDesign's XML capabilities, as we will see.

The database programmer and the InDesign layout person provided input on how they viewed the content, how they worked with it, and what problems they found when interchanging the content between the two applications. The editorial staff for the catalog also contributed input regarding how they reviewed and made corrections to the catalog during the publishing process.

Data-like Content Example: The Course Description XML

The data table that contained the course descriptions was one of the largest in the database. Hundreds of course descriptions were managed in it, containing data in a regular format, which we can demonstrate with a simple table:

Course major	Course number	Course name	Course credits	Course description	Notes
Accounting	ACC 101	Accounting Principles I	4	Basic principles of financial accounting for the business enterprise with emphasis on the valuation of business assets, measurement of net income, and double-entry techniques for recording transactions. Introduction to the cycle of accounting work, preparation of financial statements, and adjusting and closing procedures. Four class hours.	Prerequisite: MTH 098 or MTH 130 or equivalent.

Database fields for course descriptions

In InDesign, we wanted the content to look like this:

Accounting

ACC 101 Accounting Principles I 4 Credits

Basic principles of financial accounting for the business enterprise with emphasis on the valuation of business assets, measurement of net income, and double-entry techniques for recording transactions. Introduction to the cycle of accounting work, preparation of financial statements, and adjusting and closing procedures. Four class hours.

Prerequisite: MTH 098 or MTH 130 or equivalent.

There are four InDesign Paragraph styles defined for this content:

Course Descriptions—Major: The heading for the Major under which the course falls

Course Descriptions—Name: The bold text for the course number, official name and credits awarded, in a single line

Course Descriptions—Text: The normal text for the description of the course, as a paragraph

Course Descriptions—Footnote: The italics for the footnote, which is prerequisites, limitations on registration, required approvals, etc. There could be more than one paragraph of footnotes for a course.

Note: Naming all the paragraph styles with the same beginning keeps them together in the InDesign paragraph styles palette.

Example of formatted XML output for course descriptions

Data exported as XML

When we exported the course description content from the database, we combined a few of the data fields (the course name and number and credits became a single element with tabs separating the values), to align better with what the InDesign layout would be. (We knew that if we wanted to bring this content back into the database after making edits in InDesign, we would have to re-separate these values into 3 fields, with the tabs acting as a separators. However, our goal is to update the database and re-export if necessary, so that the database holds the definitive version of the course information. Editing in InDesign was a secondary concern.) This is the way the elements of a course description were written in our XML:

```
<CourseDescription_Major>Accounting</CourseDescription_Major>
<CourseDescription_Name>ACC 101      Accounting Principles I    4
Credits</CourseDescription_Name>
```

```
<CourseDescription_Text>Basic principles of financial accounting for the business enterprise with
emphasis on the valuation of business assets, measurement of net income, and double-entry techniques
for recording transactions. Introduction to the cycle of accounting work, preparation of financial
statements, and adjusting and closing procedures. Four class hours.</CourseDescription_Text>
```

```
<CourseDescription_Footnote type="prereq">Prerequisite: MTH 098 or MTH 130 or
equivalent.</CourseDescription_Footnote>
```

Sample XML structure based on database fields

The "notes" content from the database entry for a course was named `<CourseDescription_Footnote>` so that it could be recognized as a specific type of note. `<CourseDescription_Footnote>` was given an attribute named "type," which is used generally as an indication of a prerequisite for the course, if there is one. This allowed for notes that pertain to prerequisites to be searched for within the XML content.

Modeling the structure for the import XML

A simple DTD for the course descriptions data was generated from the XML that we extracted from the database. All of the course description elements are wrapped together in a root element named "CourseDescriptions":

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- DTD generated from database XML content using XML Spy -->
<!ELEMENT CourseDescriptions (CourseDescription_Major* | CourseDescription_Name* |
CourseDescription_Text* | CourseDescription_Footnote*)+>
<!ELEMENT CourseDescription_Major (#PCDATA)>
<!ELEMENT CourseDescription_Name (#PCDATA)>
<!ELEMENT CourseDescription_Text (#PCDATA)>
<!ELEMENT CourseDescription_Footnote (#PCDATA)>
<!-- ATTENTION: CourseDescription_Footnote
      type CDATA #REQUIRED -->
```

Simple DTD for course descriptions

Note: We could have wrapped the basic structure of each course with all its fields inside an element named `<CourseDescription>`, but InDesign does better with XML that doesn't have many levels of content hierarchy. So we arbitrarily made this structure simple to make it easier for the InDesign layout person.

With a simple DTD and an understanding of the basic XML structure and the paragraph styles that we are going to use in InDesign, our prep work for this import is done. We'll dive into the details of the import and paragraph styles

mapping later in this Short Cut. (If you want to understand DTDs better, see O'Reilly's *XML Elements of Style* or search for "XML DTD basics" online.)

Topical Content: The Handbook XML

We needed to reverse the process when we wanted to export the XML from InDesign to put into the database. We started by looking at the content in InDesign, thought about how we were going to store it in the database, and designed the XML markup that would achieve our goals.

Evaluating the handbook text for structure

The text in the handbook was organized into topics, such as:

- Rights and Freedoms of Students
- Code of Conduct
- Grievance Procedure
- Parking Regulations
- Alcohol and Drug Policies

Some of these topics included many subtopics, some included procedures, and some included reference tables or illustrations. Compared with the database content, this content was much more free-form and harder to predict, so the XML structure would have to be more generic.

To make XML that would be useful for the particular processes of this college, we determined that we would make each main text flow into an XML file, which would be changed into a rich text blob in the database (because that would be the most editable form of the content for the future editing cycles).

Modeling the structure as a set of topics

The content was usually edited as a single "story" or text flow in InDesign. Some of these were small and simple enough to be made into a very shallow structure: a <Story> element that contained an optional <IntroBlock> element, at least one <SectionHead>, some <SubsectionHead>s, <Subhead>s, and <para>s and optional <listitem> and <table> elements. The most complex content needed to accommodate a number of topics inside a story, with the same basic headings, paragraphs, lists and tables inside a topic. We decided that content should generally be no more than three levels deep inside a story or a topic.

Our basic structure for these types of content is captured in a tree diagram:

Story

@name

^ IntroBlock

para

Section Head

SubSectionHead

Subhead

^ keyword

para

^ keyword

^ listitem

^ Table

^ Cell

^ keyword

topic

@title

para

^ keyword

^ listitem

^ keyword

^ Table

^ Cell

Levels of XML content (hierarchy) as a tree diagram

A few elements and attributes were designed to help us manage or search the content after export. There is an attribute "name" for a <Story> element to give us a handle on the kind of information contained in a Story, such as "Career and Transfer Programs, Certificates and Advisement." A similar attribute, "title" was used on a <topic> element, so that we could identify the information in a topic, even if it did not have a heading to display. The <keyword> element could be used inside a <Subhead> or <para> element.

We did not have to be very rigorous in developing our structure. We selected names that were quite generic, and flattened out structures where we didn't think "wrapper elements" would be necessary. For example, we did not wrap a set of <listitem> elements in a <list> element. While that is common in HTML, it would be unnecessary in tagging text in InDesign, where we want the closest match we can get between the incoming elements and the number of paragraph styles that we

will use. (Adobe has a similar strategy in regard to tables, having decided to dispense with <Row> and just use <Table> and <Cell> elements.)

Note: We used names of existing paragraph styles for a few elements, and kept their capitalization, such as <SectionHeading>, while we lowercased all the more generic elements, such as <para>. This made it easier to remember what element names originated from the InDesign layout.

With this basic structure converted into a DTD, we were ready to start marking up InDesign content as XML and validating it.

Iteration and refinement

We didn't get the structure that we used on the first try. The first versions of the XML structure were more granular (had more little elements within the <topic> and the <para> level of structure) and had many more "wrapper elements." We tested by importing XML with various structures and different settings of the Import Options dialog, to see what results we got in InDesign. If we didn't like the results, we changed the structure and tried again. When we were finished iterating, I generated DTDs from our final XML, and used that for validating the content.

Note: In the chapter about validation and DTDs, you will see why I prefer to go with the minimum of structural rules, and develop DTDs after creating working examples of content (if you are "rolling your own" DTD).

In our case, we only had to be sure that one InDesign layout person and one database developer would be able to understand how to create, manage and interchange a specific set of content elements.

Net results: vast improvements in understanding and speed

We had a lot of successes with our project. One of the most significant was a somewhat improved understanding of the database by the publishing group, and much greater understanding by the database team of the publishing process. Because the bulk of the work was going to be passing content from the database to the publishing application via XML, the database programmer was intimately involved in understanding how the layout person perceived the content, and what tasks he needed to perform with the content.

Besides improved comprehension between the functional groups, there was also a very important improvement in time to delivery for the layout person. He was given a brief tutorial on XML import, and adjusted Paragraph names before importing the XML. Thereafter, where once he had spent days (literally) marking up the 130 pages of plain text paragraphs, he now could import all of the content in a few minutes, watch it auto-format itself as it came in, and then page through it, applying column and page breaks as needed. The estimated time saving for layout labor of the 130 pages was about 80 percent.

The text that was exported as XML from InDesign was marked up by an outside vendor, to minimize impact on the production cycle for the catalog. The database programmer was again a critical person in the success of the process change, by figuring out how to get the database (which did not store XML natively) to import XML and achieve a useful, editable set of new content pieces within the database. (This database could be coaxed to import the XML for each story and process it to generate a rich text field in the database, with the "name" attribute of the <Story> element providing the means to classify the story and its topics. If you are working with a database that can store native XML, your process will be simpler.)

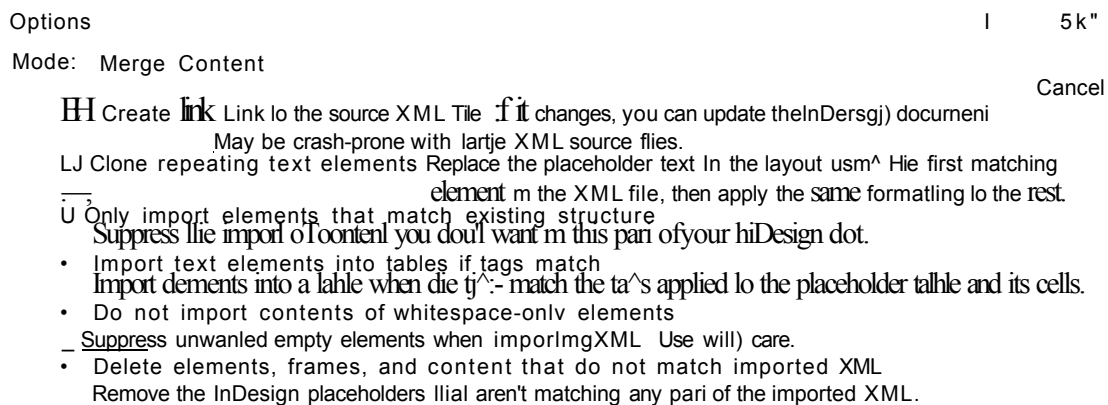
Our project was stretched out over a year's publishing cycle, and we had regular meetings and a Wiki to help track progress and document the project. I consider it a successful pilot of the processes that I am describing in this Short Cut. The process has been in use for five years (as of 2010), and the college's developers have been able to extend and adjust the process without difficulty.

Importing XML

There are several ways to work with Adobe's XML import capabilities. We'll start with the process that has been documented in a short Adobe tutorial, **Format XML in an InDesign Template**, which you can download (search online for it by title) and print for reference. Also, see the Adobe video tutorials at <http://tv.adobe.com>.

Doing It Adobe's Way: The Placeholder Approach

XML Import Options



The XML Import Options dialog with annotations

Note: if you import XML without any preexisting maps for paragraph styles, all the imported content will look like the default style that you get when you make a new paragraph without applying a style to it (the Basic Paragraph Style).

Adobe expects you to create a model in your InDesign document for your XML content.¹ The model is made of placeholders, which are XML elements that indicate what the structure of the incoming XML will be and how you want it to look.

This is a very sensible approach when you are starting out with XML and want to get a feel for how the imported XML will be formatted in InDesign.

Let's walk through the steps of the placeholder approach, using the course description content.

¹ See **Create placeholders for repeating content** on Adobe's web site for a tutorial on placeholders. Or refer to **Format XML data in an InDesign template** for an approach that is slightly different from what is described in this **Short Cut**.

Model the XML you want

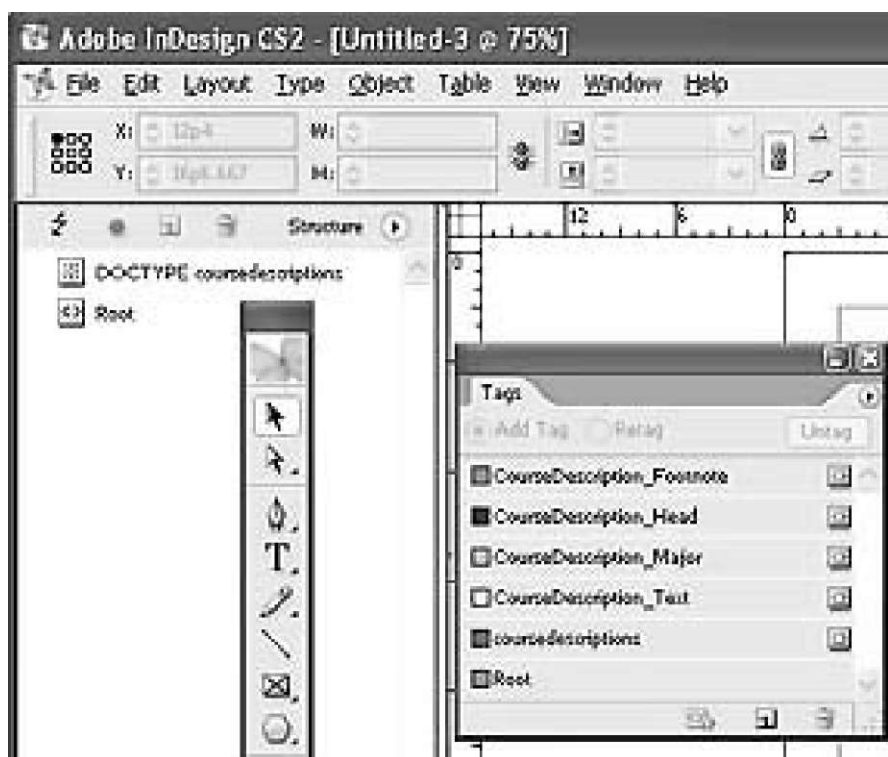
Get some structure into InDesign

The basis of your modeling will be a set of XML elements, from an existing DTD or XML file.

If you have XML based upon a DTD, start by importing the DTD into InDesign.

1. In your InDesign document, select **View -> Structure** or click on the Structure pane icon < | > in the very far left bottom corner of the document window.
2. Select **Structure>Load DTD** and then browse to select the DTD for your XML content. Click **OK** and a DOCTYPE declaration will appear at the top of the Structure pane.

Now we need to work with the Tags window, so select **Window>Tags**. Because you just imported a DTD, the Tags window is populated with the element names from your DTD.



The Tags window contains each element name (a.k.a. "tag") that is used in the DTD, in alphabetical order.

² InDesign CS2-CS5 does not provide XML schema support. You can convert a schema to a DTD with XML Spy, Oxygen Editor and other XML tools.

The Structure pane showing the DTD loaded at the top left (as a DOCTYPE declaration), and the Tags window on the right showing the element names loaded from the DTD.

If you don't have a DTD, you can load the structure directly from an XML file by importing the XML file.

1. Open the **Tags** palette from the **Windows** dropdown, then click the small arrow in the upper right of the **Tags** window to get the **Tags** menu.
2. Select **Load Tags**, and then browse to the XML file that you will use as a source for your XML import.
3. Select it and click **OK**. Element names will appear in the **Tags** window.

Create placeholders for XML elements

Get a Text frame by clicking on the **T** icon and then onto your empty document. Drag out a text frame large enough to work in.

Now you can start making your placeholder text in the text frame.

You need to choose a "wrapper" element that all of the other elements will reside within. If you imported a DTD, that will be the root element of your DTD, which in the case of my simple example is `<CourseDescriptions>`:

```
<?xml version="1.0" encoding="UTF-8"?>
<!ELEMENT Course Descriptions (CourseDescription+)>
<!ELEMENT CourseDescription (CourseDescription_Major | CourseDescription_Name |
CourseDescription_Text | CourseDescription_Footnote)+>
<!ELEMENT CourseDescription_Footnote (#PCDATA)>
<!ATTLIST CourseDescription_Footnote
    type CDATA #REQUIRED

<!ELEMENT CourseDescription_Major (#PCDATA)>
<!ELEMENT CourseDescription_Name (#PCDATA)>
<!ELEMENT CourseDescription_Text (#PCDATA)>
```

To start your placeholder XML, you will apply the root element tag.

1. Highlight the text frame and click the corresponding root tag name in the Tag palette.
2. Now, within the **View** menu, find the **Structure** item and expand it. Toggle to **Show Tagged Frames**, if **Hide Tagged Frames** is displayed. Toggle to **Show Tag Markers**, if **Hide Tag Markers** is displayed. This will provide

color-coded backgrounds on text frames and brackets around elements to help you see tags when you apply them.

3. Type the name of each element of your XML structure on a single line *in the order that they should appear in the document*. For my example, that means typing the following element names:

CourseDescriptionMajor
CourseDescriptionName
CourseDescriptionText
CourseDescriptionFootnote

4. Then, tag each line of text with the matching XML tag by clicking the corresponding name in the Tags palette.
5. Save your file.

The screenshot shows an XML editor interface. On the left, a text frame contains the following text with XML tags applied: `<CourseDescriptionMajor>`, `<CourseDescriptionName>`, `<CourseDescriptionText>`, and `<CourseDescriptionFootnote>`. The text is: "f 4 j, ilrjifm • I FT77" E h", "jj i>:>; } p p£ C (.TrOl! HfeMi / hi", "¿'lu'jibdiiVKA.Mn' M-.", "Jj. KM", "CouraeDe scription_ ftfajor", ".CourseDe scription Head", "Co in se De.scripi io&JTex t", "CourseDe scription Fcoruote". On the right, the Tags palette is visible, showing a list of XML tags: " 1", "J", "J", "J", "J", "J", "Q".

) reflects the organization of the elements and attributes of the XML. Colored brackets around each tag in the text frame (center) correspond to the color of the tag in the Tags palette (right).

Creating test XML

You can work faster if you use a small XML file for testing. Open the XML you want to import in an XML or text editor, and trim it down to just a few sets of repeating content. For our example, that would be, a `<CourseDescription_Major>` such as Accounting, and a couple of sets of `<CourseDescription>` elements, each containing `<CourseDescription_Name>`, `<CourseDescription_Text>`, and `<CourseDescription_Footnote>` elements, to make sure that the imported content has enough variation to be a good test.

Note: If you are uncertain about editing the XML file, start by saving a copy, then remove XML elements inside the root element (do not delete the root element itself), until you have just a few blocks of content containing the element structure that you want to test. Save the trimmed file and use it for your placeholder tests.

(The more complex your XML and DTD, the larger a set of text elements you will need to cover all the possibilities of your imported XML. You can, of course, try to import the entire XML file that you plan to work with if you are feeling brave.)

If you have a DTD, but don't have the actual XML you want to import, you can create an XML file from the DTD using an application such as XML Spy or Oxygen, but you will have to create at least the amount of content that I describe (several repeating blocks of content at each level of structure that you expect to repeat).

This is what I am going to import into my placeholders for testing:

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
```

```
<CourseDescriptions>
```

```
  <CourseDescription>
```

```
    <CourseDescription_Major>Accounting</CourseDescription_Major>
```

```
    <CourseDescription_Name>ACC 101    Accounting Principles I    4
```

```
Credits</CourseDescription_Name>
```

```
    <CourseDescription_Text>Basic principles of financial accounting for the business enterprise with
emphasis on the valuation of business assets, measurement of net income, and double-entry techniques
for recording transactions. Introduction to the cycle of accounting work, preparation of financial
statements, and adjusting and closing procedures. Four class hours.</CourseDescription_Text>
```

```
    <CourseDescription_Footnote type="prereq">Prerequisite: MTH 098 or MTH 130 or
equivalent.</CourseDescription_Footnote>
```

```
  </CourseDescription>
```

```
  <CourseDescription>
```

```
    <CourseDescription_Name>ACC 102    Accounting Principles II    4
```

```
Credits</CourseDescription_Name>
```

```
    <CourseDescription_Text>A continuation of the basic principles of financial accounting including a
study of partnerships and corporation accounts. The course deals with the development of accounting
theory with emphasis on managerial techniques for interpretation and use of data in planning and
controlling business activities. Four class hours.</CourseDescription_Text>
```

```
    <CourseDescription_Footnote type="prereq">Prerequisite: ACC 101 with a grade of C or higher, or
ACC 110 and ACC 111 with an average grade of C or higher.</CourseDescription_Footnote>
```

```
  </CourseDescription>
```

```
...more
```

```
</CourseDescriptions>
```

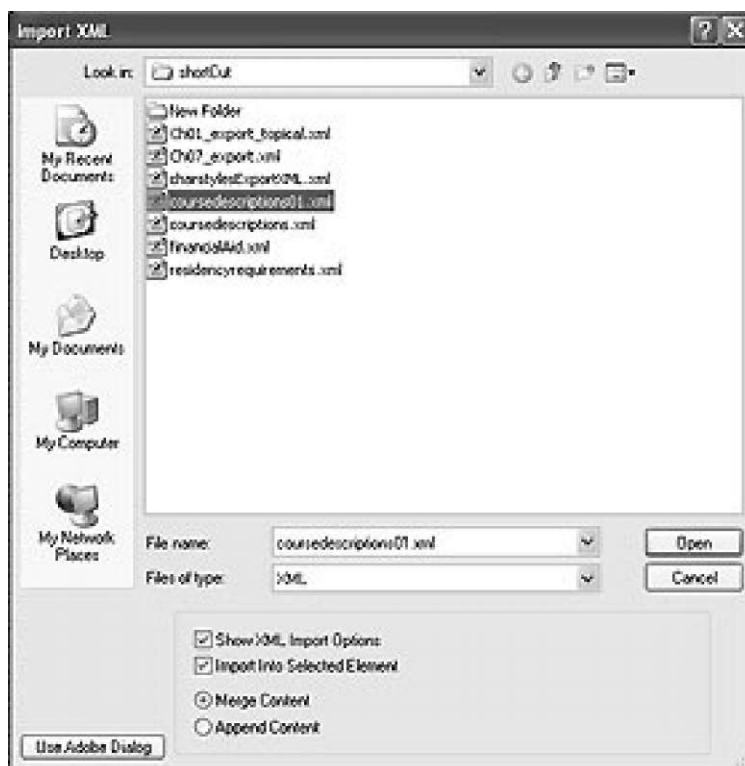
Sample XML for course descriptions import

Note: As I mentioned, we combined the values from three data fields into the single XML element named `<CourseDescription_Name>`. These values are separated by tabs. This is a convenience for print publishing, where tabs are used for better readability. If you were creating a table from the XML import, you could keep these three field values distinct by making each one an XML element and applying formatting to create table cells from the imported XML. We'll look at tables in some detail later in this Short Cut.

Importing XML into placeholders

To import the XML:

1. Select the text frame, then the File menu, and locate the **Import XML** item in the drop down menu. Select this and the **Import XML** window will appear.
2. Browse to your sample XML file and select it, then check the boxes beside **Show XML Import Options** and **Import Into Selected Item**, and the radio button beside **Merge Content**.



The Import XML dialog

3. Check the boxes for **Create link** and **Clone repeating text elements** and then click OK.

XML Jwpert OfillM?

Options

Mode: Merge Content

OK J

Cancel I

(/Ictare repeating tart element

-ZlOnty inpoft elements that match a*kthg structure

!"[jncM>t tfl*t elements nto (zfciei if t<<s iwath

LlOo not import contents of whitespate-ortY elements

- Delete elements, frsnes, jnd content that da not match mpocted MM

The Import Options dialog

Your XML content will now appear in the **Structure** pane to the left of your InDesign document pane. The text of the first part of the XML should fill the text frame on your page. It will all look like the **Basic Paragraph Style** of the InDesign document at this point.

4. Save your InDesign file to preserve its current state before you add styles.

Note: When you have a lot of XML in a file, it can become very confusing to relate where you are in the Structure pane to where the text of the element appears in the text flow. You can highlight an element in the Structure pane, then use the **Structure** pane menu (upper right corner) and select **Go to Item** to highlight the location of an XML element in the text flow.

Adding style to the XML elements

Now that you have imported the XML, you need to make it look as you desire, which means you need to assign the appropriate Paragraph style to each type of content. Fortunately, Adobe provides an easy way to apply the Paragraph styles, which you will first create based on the names of your XML elements.

1. First, in the **Paragraph** palette, create one new Paragraph style for each of the elements except the root element, naming each one exactly as the XML elements are named.
2. Give each style distinct fonts, weight, etc., so that you can easily see the differences between them when you apply them to the text in your text frame. (If you need information on creating new Paragraph styles, see the InDesign **Help** files.) Now you can map these new Paragraph styles to your XML elements.

Mapping styles to tags

You are ready to map tags to styles. At the top right edge of the **Structure** pane is a button which expands to a menu.

1. In this menu, select **Map Tags to Styles** and a dialog will open.

Map to Styles

Tag	Style	
CmjsFOrxroi on_Footr>?i?	[Not M'PI&d)	I ____<_
CtkrseOesciipTjrt;»	[No' ^jpoerIJ	<u>Cancel</u>
Cou-sfDfÇfiironi Warie	[Hal l/ipoedj	<u>Load...</u>
C w seDescj EitKri_Te* t	[Nür	• Ploie*
KxiieDesatrtoTî	[Not Mjpoedj	
<u>My. fy hire"</u>		

The Map Styles to Tags dialog, before mapping

2. Review the names of the elements and the names of your Paragraph styles. If they match exactly (upper/lower case matters!), click the button that is labeled **Map By Name**. (If they do not match, change the names of the Paragraph styles, *not* the XML tags, then use **Map By Name**).

The dialog should now show the names aligned like this:

Map Tag la Styles

c öj secesa titp> i _r iin t note H üouTEeDiicpJün¹.
 C ri,j J hi ¥: ̣. r CilmJAity C&j ieDeiCrt'ioriJ-'.
 CojieDKcptonJiafine
 1 GoLfseDeimpJai T.
 CoLfsetecrptBrß
 TtiSifiPTtSIi^ v

I CK 1

Cancel

I. L<cL ;

I MftjBfMgira :

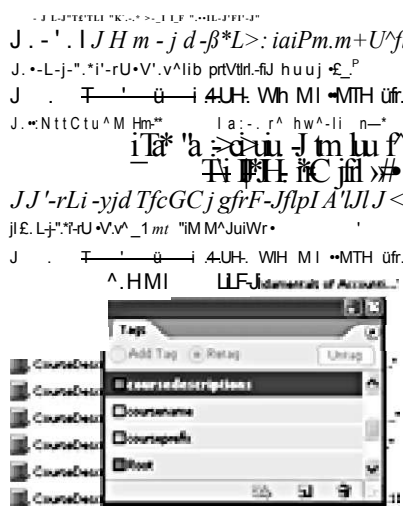
The Map Styles to Tags dialog, after mapping

Note: If the Tag names do not map when you click **Map By Name**, you can use the Style dropdown to the right of each tag name to select the matching Paragraph style. Usually, if they do not match, it is because you have used spaces in the paragraph style names (spaces are not allowed in XML element names), or capitalization is different.

3. Click OK. You will see that the text in the text frame is now styled with your Paragraph styles. (This is typically the moment when you involuntarily cheer, if indeed you do not leap from your chair.) Save your file.

i 1 y }

1111111111 K 1111111111 M 1111111111 IM I 11111111 FM I 11111111 FM I 11111111 FM I 11111111 IM 11111111



J. - / Lj-?rU-y;-c_1* 1 + ^ i j t H Nn-TihJvdvU¹
 Hu 14X KJL jH I II.
 J.'nrtfMl'pWA.Hhd 'AK Tti fAWJii IL K-j'h_j*
 J. - IA^rT-Li # 'A Stedi-üi Ilic-. iriJ
 jI. - J l-j'£'LI k-.* _F i>V' • Hu 14-rX' r_ lb I T'. h'h-.*
 Mx iri' 14X AC C J K i Mi • g' *
 M - Lj-?rU _TnE >Kp i kUh«Eil» i SUI J h-m
 # 14X AC C J K i Mi • g' *

Accowino
 ACC 101 Accounting Principles I 4 Credits
 Basic principles of financial accounting for the business enterprise with emphasis on the recording of business transactions, measurement of net income, and basic entry techniques for recording transactions. Introduction to the cycle of accounting with preparation of financial statements, and adjusting and closing procedures. Four class hours.
 Prerequisite: ACC 101 or ACC 101 with a grade of C or better.

PK pH * • .ibi"
 applied/practical approach to the operation of computerized general ledger system. Material covered and include accounts receivable, inventory management, sales training, accounts payable, and cash management. Emphasis is placed on the use of special journals, subsidiary ledgers, provides employee reference. Scheduled to be offered in the Fall Semester during the day and the Spring Semester during the evening. Three class hours.
 Prerequisite: ACC 101 or ACC 101 with a grade of C or better.

1KB* " l'jp

Phnil ' iP in-v/ar - L r ' /r Ha

Afcgial/Clrcnid
 hgggdengt

' tH' n • • IIMI n-HKThfiIM

I>rIwi.'i'ari rnp
 nr-iL Hi air H iH i m / W
 - T 4 - tManinl

tmBif n m HWG
 jurninrj
 training process has credit given for both ACC 101 and ACC 101. Students for completion either ACC 101 and ACC 101 is equivalent to ACC 101. Two class hours, one conference hour.
 Prerequisite: ACC 101 or ACC 101 with a grade of C or better.

ACC 101 Intermediate Accounting I 4 Credits
 is more analytic in treatment of accounting theory and practice, with a review and application of basic procedures. Topics include cash, receivables, inventories, plant assets, intangible assets, current and long-term liabilities, long-term debt, and financial statement presentation and disclosure. Scheduled to be offered in the Fall Semester during the day and the Spring Semester during the evening. Four class hours.
 Prerequisite: ACC 101 or ACC 101 with a grade of C or better.

ACC 101 Intermediate Accounting II 4 Credits
 is more analytic in treatment of accounting theory and practice, with a review and application of basic procedures. Topics include cash, receivables, inventories, plant assets, intangible assets, current and long-term liabilities, long-term debt, and financial statement presentation and disclosure. Scheduled to be offered in the Fall Semester during the day and the Spring Semester during the evening. Four class hours.
 Prerequisite: ACC 101 or ACC 101 with a grade of C or better.

H i Indult

v-'h H fl Oct-

• i

Imported XML, autoformatted when imported into InDesign

You can look at the imported, formatted XML in your text frame and compare it to your original XML to make sure that the mapping has applied styles as you intended.

Accounting

ACC 101 Accounting Principle;]
4 Credits

Basic principles of financial accounting for the business enterprise with emphasis on the valuation of business assets, measurement of net income, and double-entry technique* for recording transactions. Introduction to the cycle of accounting work, preparation of financial statements, and adjusting and closing procedures. Four class hours.

prerequisite: MTH OPS or MTH IW or equivalent

ACC 102 Accounting Principles II
4 Credits

A continuation of the basic principles of financial accounting including a study of partnerships and corporation accounts. The course deals with the development of accounting theory with emphasis on managerial techniques for interpretation and use of data in planning and controlling business activities. Four class hours.

prerequisite: ACC 101 with a grade of C or higher, or ACC 110 and ACC 111 with an average grade of C or higher.

Anthropology

```
<CourseDescription_Major>Accounting</CourseDescription_Major>
<CourseDescription_Name>ACC 101    Accounting Principles I    4 Credits</CourseDescription_Name>
<CourseDescription_Text>Basic principles of financial accounting for the business enterprise with emphasis on the valuation of business assets, measurement of net income, and double-entry techniques for recording transactions. Introduction to the cycle of accounting work, preparation of financial statements, and adjusting and closing procedures. Four class hours.</CourseDescription_Text>
<CourseDescription_Footnote type="prereq">Prerequisite: MTH 098 or MTH 130 or equivalent.</CourseDescription_Footnote>
<CourseDescription_Name>ACC 102    Accounting Principles II    4 Credits</CourseDescription_Name>
<CourseDescription_Text>A continuation of the basic principles of financial accounting including a study of partnerships and corporation accounts. The course deals with the development of accounting theory with emphasis on managerial techniques for interpretation and use of data in planning and controlling business activities. Four class hours.</CourseDescription_Text>
<CourseDescription_Footnote type="prereq">Prerequisite: ACC 101 with a grade of C or higher, or ACC 110 and ACC 111 with an average grade of C or higher.</CourseDescription_Footnote>
<CourseDescription_Major>Anthropology</CourseDescription_Major>
```

The formatted XML in InDesign (left) compared with the raw XML (right)

You can adjust tabs, change colors and fonts, and make whatever changes you want to a Paragraph style, and the changes will automatically be applied to every element (tag) with a name that matches the style.

You can also apply local overrides to the style of a particular piece of tagged text, and it will not affect the rest of the elements with the Tag name that are mapped to the same Paragraph style.

Now you can go whole hog with the entire XML file.

Importing the "real" XML file

Getting down to cases with our importing scenario:

1. Select the root node of your sample XML file in the **Structure** pane.
2. Delete the file. A dialog will appear asking for confirmation that you want to take this action; click **Delete**.
3. Then, **Import XML** as before, but select the real XML file you want to publish, rather than your sample file. Select **Merge** to replace the empty root element with the new XML. You can see the styled text reappear in the text frame, and a lot more XML elements will appear in the Structure pane.

Note: There may be a delay while the **Import Options** dialog processes the XML file when you import something larger than the sample file we have used so far. In the case of very large XML files, InDesign will occasionally crash. If that happens, consider breaking the XML file up into a few smaller files and **Append** each one on import, rather than using **Merge**, so that they will appear in the text flow in the order that you append them.

XML is memory-intensive. If you are short on memory (and money to upgrade your hardware) and your documents are freezing, consider making a large document into several smaller ones and putting them together with the Book feature in InDesign. See your InDesign documentation for information on Books.

If you want to see exactly where the tags appear around the text in your file, you can inspect them with the **Story Editor**. Under the **Edit** menu, choose **Edit in Story Editor**. Each piece of tagged text in the flow will appear with the angle symbols around it, color-coded to the Tag color. Notice that paragraph breaks appear after the ending angle symbols.

```
[CourseDescription_Name) Co U TSE Des C' 1 pt1 0n_N jme {/CourseDe5cription_Name
j CoureeDe5cription_Footnote) Co UTS eDeS CT 1 p 110ri_F00tn0ie ffCoureeDe5cription_Footnote
CourseDescription_Teair) CPU TSE Des C ri pt1 011 _TEX t(/CourseDe5cription_Te'it'^
```

The Story Editor showing tag markers (angle symbols) around tagged text.

Note: If you put a line break or tab inside the tag markers of an XML tag, these will be treated as normal tabs and paragraph breaks by InDesign.

If you remove the line break between two tags, the content will run together in a single paragraph.

Scroll down through the file and look at the tags. If something appears to have been tagged incorrectly, you can select it, and then click a different Tag name in

the **Tag** palette. You can also drag elements into a different order if the order is wrong (for example, moving the footnote in the illustration above to the position after the description text).

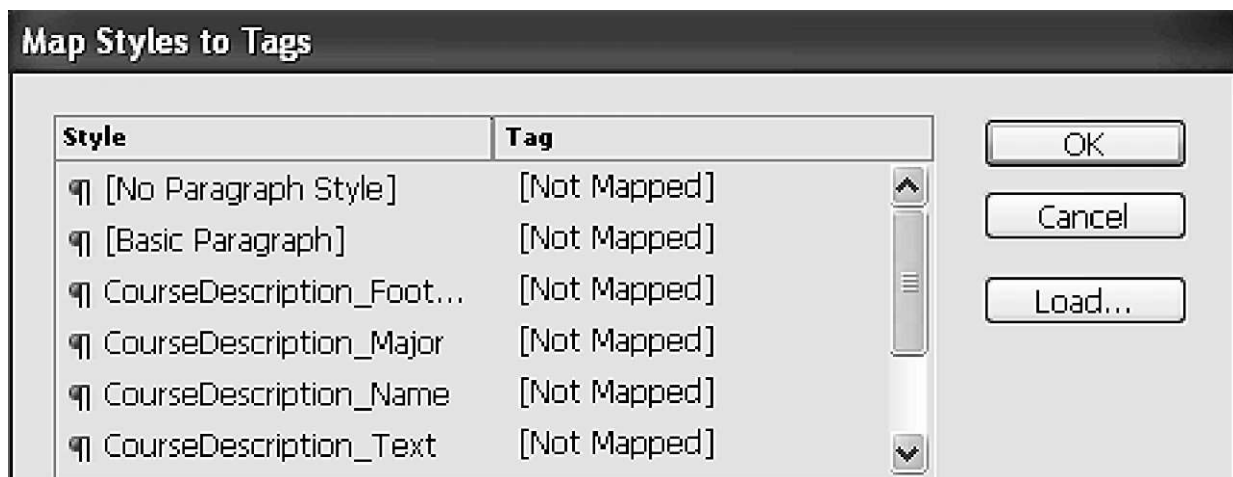
Once you are satisfied that the tagging is correct, use the **Edit** menu to switch back to **Edit in Layout**. Now you can make additional text frames and flow the text as you would normally flow any text in your InDesign layout. (If you need assistance in setting up text frames and flowing text, see the InDesign **Help** files.)

Note: Use autoflow to fill threaded frames that run from page to page. In CS2 or CS2, add pages and then thread more frames to continue the flow of XML.

With InDesign CS4 and CS5, you can use Smart Text Reflow features to add pages full of content to your document automatically.

An aside: the scary Map Styles to Tags dialog message

When you were using **Map tags to Styles**, you may have noticed that there was an option to **Map Styles to Tags**. If you choose **Map Styles to Tags**, the dialog for mapping looks very similar but you will see a message on it that may give you pause: "Note: Mapping styles to tags completely restructures the text in your document."



Note: Mapping styles to tags completely restructures the text in your document.

The Map Style to Tags dialog (warning text circled)

Yipes! What does that mean? Adobe is warning you that these mappings are going to apply *all across your current document*. If you **Map by Name** and the names

match, and you only have content for one type of XML structure, all should be well. This dialog is generally used when you *don't* have imported XML already in your InDesign file.

Note: Don't map the Basic Paragraph to any XML element if you want to be able to have non-XML content in the same document as your XML.

But what happens if you already have XML tags on your text and you now use **Map Styles to Tags**? If you use Map Styles to Tags, InDesign will change your structure to match the Paragraph Styles that you select with the dialog or map by matching names, which might cause problems. If you have differences between the Paragraph Style names and the XML tag names, your XML elements will now have names that may not match your DTD. If some of the text doesn't have a corresponding XML tag, the mapping will only be partial. In either case, this will cause your XML to be invalid, a problem you might not notice until you try to export XML out of InDesign to use for another process that depends on the DTD.

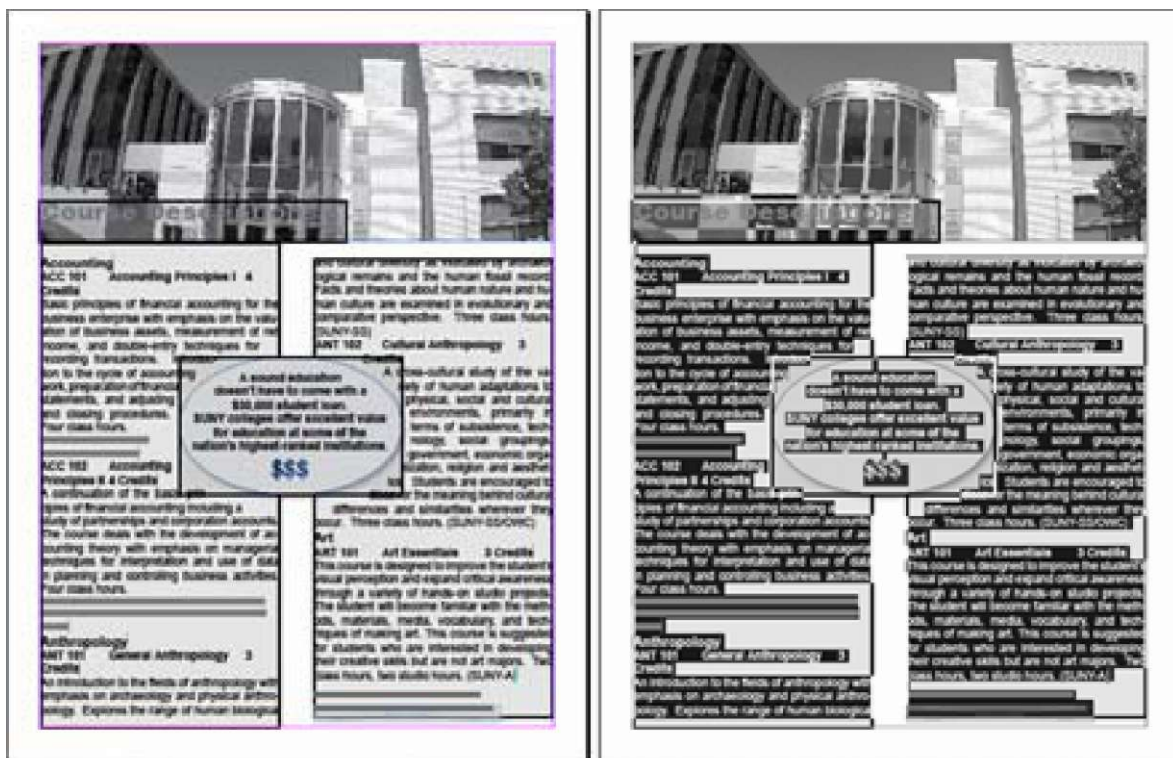
^ The rules of thumb should be: use **Map Style to Tags** for tagging text for exporting XML that you have created from InDesign. Use **Map Tags to Styles** when importing XML to create text in your InDesign document.

Mingling non-XML and XML content in a text flow

It is possible to create a text flow that contains untagged (non-XML) content along with XML. We can have the content set up as a running text flow, with all of the courses for a major occurring one after another. We can use the same XML, but include some non-XML elements as well.

Note: The InDesign documentation describes a method for including text between XML elements that form repeating blocks. For example, you could make a set of labels that would precede each element in a block. The best way to understand this is try it out with small sample XML files, following the instructions in the Help for using **placeholders** with XML.

You should use the repeating blocks techniques whenever your XML content has nested elements in consistent, orderly structure. You would *not* use repeating blocks when the XML content is variable in structure.



Using Select All to see where the text flow appears on the page (right)

When a text flow contains both XML-tagged and normal text, you can see this by placing the cursor in the text flow and then choosing **Edit -> Select All**. The text in the flow will be reverse highlighted (as shown in the preceding illustration), so it is easy to see where the flow is on the page. You can switch to **Edit>Edit in Story Editor**, to see that the normal text has no tag markers around it.

Course Descriptions

A sound education doesn't have to come with a \$30,000 student loan. SUNY colleges offer excellent value for education at some of the nation's highest-ranked institutions.

G3II

j Course Descriptions ^ Course Descriptions ^ Accounting Principles I {Course Descriptions_Major
^ Course Descriptions_Name) ACC 101 Accounting Principles I 4 Credits (Course Descriptions_Name ^
courseDescription_Text) Basic principles of financial accounting for the business enterprise with
emphasis on the valuation of business assets, measurement of net income, and double-entry
techniques for recording transactions. Introduction to the cycle of accounting work,
preparation of financial statements, and adjusting and closing procedures. Four class
hours. {Course Descriptions_Text

A mixed text flow of untagged and tagged content

The untagged text will be included in the XML if you export the whole mixed text flow. Manual line breaks and paragraph breaks (which may need to be stripped out later, depending on your processes) will be included in the XML output file. See **Bad Characters** in the section **What InDesign Cannot Do (or Do Well) with XML** for more information.

Note: If you use a DTD, and the DTD doesn't permit plain text within the root element, or between elements, your XML will not validate. So you may have to get rid of the untagged text with a subsequent process such as an XSL transforms. Or an XSLT can be used to make the untagged lines of text in the <Story> element into individual XML elements, such as <h1> and <p>. See the sections on **Validating XML in InDesign** and **Advanced Topics: Transforming XML with XSL**.

Note: Sometimes you need more than just the Export with XSLT features of InDesign to get the final XML result you want. You would start by using an XSLT on export, then use a separate process to do more transformation on the XML content until it conforms to your target DTD and validates.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
```

```
<Root>
```

```
  <Story>Course Descriptions
```

A sound education doesn't have to come with a \$30,000 student loan.

SUNY colleges offer excellent value for education at some of the nation's highest-ranked institutions.

```
<CourseDescriptions>
```

```
  <CourseDescription>
```

```
    <CourseDescription_Major>Accounting</CourseDescription_Major>
```

```
    <CourseDescription_Name>ACC 101    Accounting Principles I    4
```

```
Credits</CourseDescription_Name>
```

```
    <CourseDescription_Text>Basic principles of financial accounting for the business enterprise with emphasis on the valuation of business assets, measurement of net income, and double-entry techniques for recording transactions. Introduction to the cycle of accounting work, preparation of financial statements, and adjusting and closing procedures. Four class hours.</CourseDescription_Text>
```

```
    <CourseDescription_Footnote type="prereq">Prerequisite: MTH 098 or MTH 130 or equivalent.</CourseDescription_Footnote>
```

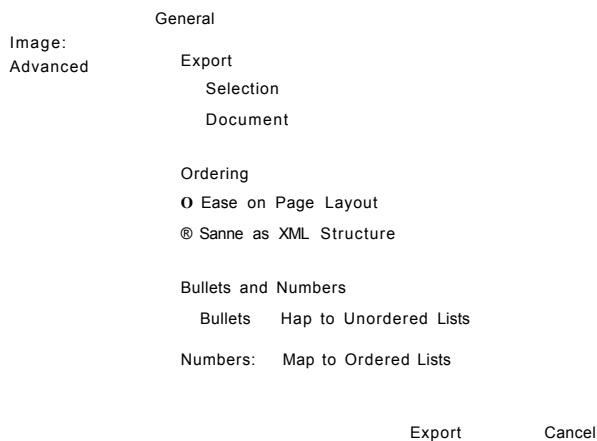
```
  </CourseDescription>
```

A sample of XML containing unmarked text (highlighted text between the Story and Course Descriptions elements)

Exporting XHTML when XML is in your InDesign file

Besides "Export as XML", exporting XML-tagged content as XHTML from InDesign CS4 or CS5 is another option. In CS4, it's an either-or choice: File -> Export for Dreamweaver (as XHTML) or -> Export as XML. In CS5, you have File > Export For > Dreamweaver and then in the XHTML dialog, you have the option to export XHTML in Order -> Same As XML Structure.

What does that mean?

XHTML Export Options**The XHTML Export Options dialog—new choices**

Actually, it's a tip about what the XHTML export generally does - all the content you have selected for exporting as XHTML comes out from the top-left to the bottom-right corner of your spread in X-Y coordinate position. If you select content that is tagged in XML, then in CS5 you can choose to use the XML order as the way your content will come out in XHTML.

For example, when the page spread includes an image in upper-left corner:

<pre><body> <div id="table-list-export-as- html-recipes"> <div class="image"> </div> etc.</pre>	<pre><body> <div id="table-list-export-as- html-recipes-xmlorder"> <div class="story"> <table id="table- 1"><tr><td>cell 1</td><td>cell 2</td></tr> etc.</pre>
Using "Base on Page Layout": first nested div contains an image.	Using "Same as XML Structure": first nested div contains a table.

Results of different XHTML Export Options

This can be very important for saving time in web page development. If you use the standard XHTML export, and the resulting content is not in the sequence that you want, you will have to drag content around in your HTML editor until you have it where you want it. But if the XML order is what you want, if you use the "Same as XML Structure" in CS5, you can get the exact sequence of content that you want for your web page.

Doing It Your Way: Using the Options for Your Own Process

Import XML using only Merge, no other import settings

You can use **Import XML Options>Merge Content** without any other Import Options. **Merge Content** replaces the contents of the selected text frame or structure element with the incoming XML content. If you have mapped tags to paragraph styles, the styles will be applied to the XML content as it is imported. Otherwise, the XML content will all look like your Basic Paragraph default style.

If you do not select a text frame when using **Merge**, the XML content replaces the Root element in the **Structure** pane. It will not flow into text frames. You can then drag elements from the **Structure** pane into text frames on your page to place the imported XML as you want it. If you have XML elements in your content that are not for the end user (such as metadata tags about versions or authors that only are useful to the editors and layout people), these will be included in your XML structure. If you do not drag these into text frames in your layout, they will not appear in the text flow. However, they will increase the number of elements you have to scroll through in the **Structure** pane. (To exclude these nonessential XML elements when importing XML in CS3, see **Only import elements that match existing structure.**)

Using Append

If you want to place XML content at the end of an existing text flow, use **Append** instead of **Merge**. **Append** always places the new XML content at the end of the text flow, but it does *not* replace existing content when the XML is imported. You can **Append** several XML files in the order that you want them to appear in the text flow. If you do not use **Create Link** when you import and **Append**, the XML files will all become part of one XML structure, but you will no longer have a way to manage them independently within InDesign.

Linking to external XML files

InDesign treats an imported, linked XML file as it does other external files (such as image files) that are linked when imported. Be aware that linked files increase the chances of unanticipated events (crashes), even as they provide an easy way to make sure the XML in InDesign is synchronized with the source XML.

External updates on Open and using the Links palette

If you close and reopen an InDesign file with linked XML, you will get a dialog warning if the linked file is not available (has been moved or renamed since it was linked). You can navigate to a new file or a renamed file and re-link by using the **Links** palette to refresh the XML inside InDesign. Select **Window>Links**, then click the name of the XML file in the list of linked assets. Click the **Relink** icon on the bottom edge of the **Links** window and select the file to which you linked.

If you use **Create link** in conjunction with **Append**, you can update appended parts of the imported XML structure independently.

You can choose not to link when importing XML, or unlink an XML file from its source later, if the import is a "one time" process to start a document, and you don't anticipate needing to keep synchronized with the source XML.

Creating text flows for the imported XML

Once you have imported the XML, if you did not select a text frame when importing it, you can play with the layout components and organize how text will flow from one frame to another.

For a single text flow, drag the root element of your XML from the **Structure** pane into a text frame.

If you want part of the XML content, such as an introduction, to appear in a separate frame, create another text frame and drag the introduction elements into it from the **Structure** pane. Then, drag the main content into its own text frame.

The importance of "document order" for imported XML

InDesign imports XML elements in the order in which they appear (this may seem self-evident, but there are ways to apply XSL in conjunction with XML to change the sequence order of elements). *A structure that occurs only once in your XML file cannot appear multiple times in the InDesign layout.* So, if you want a block of elements to appear multiple times, it should be repeated in the XML.

An example of this would be a Caution block, which might have the same wording every time it is used. It would be more efficient if you could drag the same structure into an anchored text frame every time you need a Caution, but InDesign will only present the structure where you first place it.

Note: If you drag the XML that you want to repeat into a frame on a Master page, it can "repeat" for each page that uses that Master layout, but it will not be related to the order of other XML elements in the document.

Rearranging XML elements in the structure pane

You can drag and drop elements in the Structure pane to rearrange them. Select the structure you want to move (a single element or a set of elements) and drag them into the new position.

Note: It is a bad idea to rearrange XML that you have imported as *linked*. There is a high likelihood of a bad crash. If you must rearrange structure in a linked XML file that you have imported, it is best to edit the file in the XML outside of InDesign. In the **Link** window, select the linked XML file, then click the "pencil" button to **Edit Original**. Depending on your settings, another application such as Dreamweaver may open, or you may have to select an application to use for XML editing. If you have nothing else, Notepad will work, but I recommend an XML-aware editor such as Oxygen or TextEdit 2.

Fixing up structure in the Story Editor

Just as in the **Structure** pane, you can rearrange XML elements in the **Story Editor** view. Make sure your text cursor is inside the flow and within an XML element, then select **Edit>Edit in Story Editor**. Hold down the mouse button and drag across one or more XML elements from the starting tag marker to the ending tag marker, then hold down the mouse button and drag the elements to the new position.

The benefit of editing in the **Story Editor** is that you can see the tag markers clearly, and thus work on deeply nested elements in high detail. The drawback is that you can't expand and collapse the XML structure, so if you need to move some XML a distance from its current position, it can be hard to tell when you are getting to the location where you want to place the elements.

^ Rule of thumb: rearrange elements in the **Structure** pane when you need to work with a large section of the XML file. Use the **Story Editor** view if you need to move XML elements precisely and you don't have to drag them very far.

Note: As noted in the section *Rearranging XML elements in the structure pane*, it is a bad practice to rearrange elements in a *linked* XML file.

You can use the Structure view to find an element that you want to edit. Open the **Structure** pane menu, and select **Go to Item**, then select **Edit>Edit in Story Editor**. The **Story Editor** will open the XML file at the location of the selected item (element).

One nice thing about the tag markers in **Story Editor** is that you cannot accidentally remove a starting or an ending tag marker from an element. The markers are artifacts of the display, not part of the text. So, you are protected from creating invalid XML that doesn't have matching start and end tags.

One bad thing about the **Story Editor** is that it is easy to mangle the visual appearance of the text in the flow without noticing. Within an element, you can use tabs, manual line breaks, or new paragraph breaks. These will then show up in the text flow when you switch back to **Edit in Layout** view. In some cases, this may force your XML to overflow the text frame. You won't be able to see where the end of the XML text flow is unless you extend the text frame or make a new text frame and connect it to the frame with the overflow. (If you need help with understanding frames and overflows, see the InDesign **Help** topics.)

It is good practice to switch back to **Edit in Layout** view frequently if you are editing in the **Story Editor**, just to make sure that you are not going to have a lot of layout problems to fix when you are finished editing the XML.

Understanding InDesign's XML Import Options

When you first encounter the **XML Import Options** dialog, you may have difficulty deciding which options to use. To start with, you should know that in InDesign terminology, an "element" can be a design or layout component such as text frame, paragraph, or image, rather than an XML <element>. A "tag" in InDesign terms refers to a piece of content marked up with XML element name. This table should help you use the **XML Import Options** dialog.

If your XML...	then choose this Import Option.	<i>Comments</i>
should create an updated InDesign file when the source XML is changed...	Create link	<i>If you use this option, you can update the XML within InDesign using the Links palette, just as you would update an image or other linked file. Adobe warns that unexpected results can occur if the updated XML structure is different from what was originally imported into InDesign. Caveat emptor.</i>
contains repeating blocks of elements that you always want to format in the same way...	Clone repeating text elements	<i>This option is best for fast auto-formatting of XML extracted from databases, or other repeating XML structures that need consistent layout. Use with placeholders that model the exact sequence in which you want to lay out the XML content.</i>
contains elements that are not needed in your current InDesign document...	Only import elements that match existing structure	<i>In this case, "elements" does seem to mean XML elements, but "existing structure" refers to the tags you have set up within InDesign, not to the structure of the entire incoming XML file. Use this setting to keep the XML import free of extraneous elements such as metadata that you didn't map to paragraph styles.</i>
should create a table layout in your InDesign document... (If you want to use this capability without it being obvious, make a table without any visible borders.)	Import text elements into tables if tags match	<i>If you have set up a table in your InDesign document and mapped tags to the table structure, imported XML will flow into the table cells. This is very powerful for formatting large tables, but may not work as expected if you need nested tables. Also, InDesign provides ways to make headers and footers for tables that run across multiple text frames, which can be used with XML import. See the separate discussion of tables elsewhere in this Short Cut.</i>

If your XML...	then choose this Import Option.	Comments
should flow around untagged elements in the InDesign document (labels, images, pulled quotes, etc.)...	Do not import contents of whitespace-only elements	<p><i>The single most confusing option, this keeps untagged (non-XML) text in a textflow from being overwritten by incoming XML elements. In InDesign terms, a whitespace-only "element" does not show up in the structured pane, but is visible in the Story Editor view. It is generally a tab, line break, or space(s). Best understood by testing.</i></p> <p><i>This option will insert labels when cloning repeating blocks created with the placeholders technique; it preserves untagged elements at the point where they appear in the textflow.</i></p>
contains optional structures...	Delete elements, frames and content that do not match imported XML	<p><i>If you are creating page layouts with tagged text frames that are not always needed on every page of similar layout, this option will remove such text frames when the XML is imported and a text frame is not needed. Adobe provides an example of a sidebar Notes text frame; when there is no corresponding Notes XML on a page, the text frame is removed from the page when the XML is imported.</i></p>

The table above is based on "Adobe InDesign CS2 and XML: A Technical Reference, " available from Adobe's web site.

Table of XML Import Options

Using "Clone repeating text elements"

Quick test: If you have created placeholders for your XML elements, select the placeholders, choose **Merge**, and import the entire XML file *without* checking the **Clone repeating text elements** option. How much of your XML file was imported?

It seems that InDesign expects you will have a repeating type of content structure, identifiable by a "wrapper" element that contains other elements. For example, the course descriptions content has a <CourseDescription> element containing the name, description, and prerequisite content. Without the wrapper <CourseDescription> element, InDesign will not properly apply the "clone" to the incoming XML when you select the **Clone repeating text elements** Option.

Note: The need for the wrapper element caused a change in the format of the original XML course descriptions content (as described in the case study at the start of this Short Cut). It is not unusual to have to iterate on XML development processes until you achieve the desired appearance for imported XML.

To clarify, here are examples:

Wrapped element structure	Unwrapped element structure
<pre><CourseDescription <CourseDescription_Name>ACC 101 Accounting Principles I 4 Credits </CourseDescription_Name> <CourseDescription_Text>Basic principles of financial accounting for the business enterprise with emphasis on the valuation of business assets, measurement of net income, and double-entry techniques for recording transactions. Introduction to the cycle of accounting work, preparation of financial statements, and adjusting and closing procedures. Four class hou rs. </CourseDescription_Text> <CourseDescription_Footnote type="prereq">Prerequisite: MTH 098 or MTH 130 or equivalent.</CourseDescription_Footnote> </CourseDescription></pre>	<pre><CourseDescription_Name>ACC 101 Accounting Principles I 4 Credits </CourseDescription_Name> <CourseDescription_Text>Basic principles of financial accounting for the business enterprise with emphasis on the valuation of business assets, measurement of net income, and double-entry techniques for recording transactions. Introduction to the cycle of accounting work, preparation of financial statements, and adjusting and closing procedures. Four class hours.</CourseDescription_Text> <CourseDescription_Footnote type="prereq">Prerequisite: MTH 098 or MTH 130 or equivalent.</CourseDescription_Footnote></pre>
<p><i>Result: Clone repeating text elements</i> will work properly if placeholders were set up; every repeating structure will be formatted identically.</p>	<p><i>Result: Clone repeating text elements</i> will only import the first structure block of the XML. Subsequent blocks will not be imported or formatted.</p>

Comparison of wrapped and unwrapped elements used as placeholders

When **Clone repeating text elements** is used as Adobe intends, it works beautifully. (I successfully created many pages of content with this option).

Importing only elements that match structure

Sometimes there is part of the XML file that you don't want to have in the printed version of the content. The most likely situation is regarding XML elements that describe the document itself, its creator, purpose, approvals, etc., commonly referred to as "metadata." If you create placeholders for all the elements that you want, but don't create placeholders for the unwanted XML elements, you can use this option to exclude them from the import.

For example, in the following XML file section, the highlighted text is not in the placeholder structure in the InDesign document.

```

<CourseDescriptions>
<metadata><creator>Hoskins</creator><createDate>03012007</createDate><note>used for table layout
(table tagged as &lt;CourseDescriptions&gt;)</note><note>placeholder text is named with element
names for clarity; paragraph style names may be different, so not using Map Tags to Styles with Map by
Name checkbox.</note></metadata>
<CourseDescription_Major>Accounting</CourseDescription_Major>
<CourseDescription>
<CourseDescription_Name>ACC 101    Accounting Principles I 4 Credits</CourseDescription_Name>
<CourseDescription_Text>Basic principles of financial accounting for the business enterprise with
emphasis on the valuation of business assets, measurement of net income, and double-entry techniques
for recording transactions. Introduction to the cycle of accounting work, preparation of financial
statements, and adjusting and closing procedures. Four class hours.</CourseDescription_Text>
<CourseDescription_Footnote type="prereq">Prerequisite: MTH 098 or MTH 130 or
equivalent.</CourseDescription_Footnote>
</CourseDescription>

```

XML example including metadata elements

Because the metadata tags do not match the placeholder elements that were set up, they can be excluded using the **Import only elements that match structure** Option. So the result should be only this in the imported XML:

```

<CourseDescriptions>
<CourseDescription_Major>Accounting</CourseDescription_Major>
<CourseDescription>
<CourseDescription_Name>ACC 101    Accounting Principles I 4 Credits</CourseDescription_Name>
<CourseDescription_Text>Basic principles of financial accounting for the business enterprise with
emphasis on the valuation of business assets, measurement of net income, and double-entry techniques
for recording transactions. Introduction to the cycle of accounting work, preparation of financial
statements, and adjusting and closing procedures. Four class hours.</CourseDescription_Text>
<CourseDescription_Footnote type="prereq">Prerequisite: MTH 098 or MTH 130 or
equivalent.</CourseDescription_Footnote>
</CourseDescription>

```

Imported XML, metadata excluded using the "Import only elements that match structure" option

If the imported XML file is linked, then the XML in the original file will still contain the elements that weren't imported. They just won't be in the InDesign document structure. So, don't expect to see them if you *export* the XML later.

Note: if you plan to validate XML, be careful not to exclude elements that are *required* by the DTD.

Avoiding overwriting text labels in the placeholder elements

In InDesign, a "whitespace-only" element is one that contains a tab, line break, or space(s), and if you find that difficult to understand by reading these words, you're not alone. The **Do not import contents of whitespace-only elements** Option could hardly be more obtuse. In the *Adobe InDesign CS2 User Guide*, the description of this option reads:

Leaves any existing content in place if the matching XML content contains only whitespace (such as a return or tab character). Use this option if you've included text between elements that you want preserved. For example, when laying out recipes generated from a database, you might add labels, such as "Ingredients" and "Instructions." As long as the parent element that wraps each recipe contains only whitespace, InDesign leaves the label in place.

I'm sure that clarifies how to use this option, but just in case it doesn't, let's break it down by example.

Here is some XML placeholder content with labels inserted between some of the elements:

```
<CourseDescriptions>
<Course Description
Major: <CourseDescription_Major>CourseDescriptions_Major</CourseDescription_Major>
<CourseDescription_Name>CourseDescriptions_Name</CourseDescription_Name>
<CourseDescription_Text>CourseDescriptions_Text</CourseDescription_Text>
    Note:
<CourseDescription_Footnote>CourseDescriptions_Footnote</CourseDescription_Footnote></CourseDe
scription>
</CourseDescriptions>
```

Placeholder XML elements

Here is the XML structure that we are going to import:

```
<CourseDescriptions>
    <CourseDescription_Major>Accounting</CourseDescription_Major>
    <CourseDescription>
        <CourseDescription_Name>ACC 101    Accounting Principles I    4
Credits</CourseDescription_Name>
```

³ *Adobe InDesign CS2 User Guide* and *Adobe InDesign CS3 User Guide* are available from the Adobe Store online.

```
<CourseDescription_Text>Basic principles of financial accounting for the business enterprise with
emphasis on the valuation of business assets, measurement of net income, and double-entry techniques
for recording transactions. Introduction to the cycle of accounting work, preparation of financial
statements, and adjusting and closing procedures. Four class hours.</CourseDescription_Text>
```

```
<CourseDescription_Footnote type="prereq">Prerequisite: MTH 098 or MTH 130 or
equivalent.</CourseDescription_Footnote>
```

```
</CourseDescription>
```

```
</CourseDescriptions>
```

Actual XML content to replace placeholders

If we use the **Do not import contents of whitespace-only elements** option when we import XML content into an InDesign template document with our placeholder text, the result of the import will be labeled entries in the final XML, like this:

```
<CourseDescriptions>
```

```
Major: <CourseDescription_Major>Accounting</CourseDescription_Major>
```

```
<CourseDescription>
```

```
<CourseDescription_Name>ACC 101 Accounting Principles I 4 Credits</CourseDescription_Name>
```

```
<CourseDescription_Text>Basic principles of financial accounting for the business enterprise with
emphasis on the valuation of business assets, measurement of net income, and double-entry techniques
for recording transactions. Introduction to the cycle of accounting work, preparation of financial
statements, and adjusting and closing procedures. Four class hours.</CourseDescription_Text>
```

```
Note:
```

```
<CourseDescription_Footnote type="prereq">Prerequisite: MTH 098 or MTH 130 or
equivalent.</CourseDescription_Footnote>
```

```
</CourseDescription>
```

```
</CourseDescriptions>
```

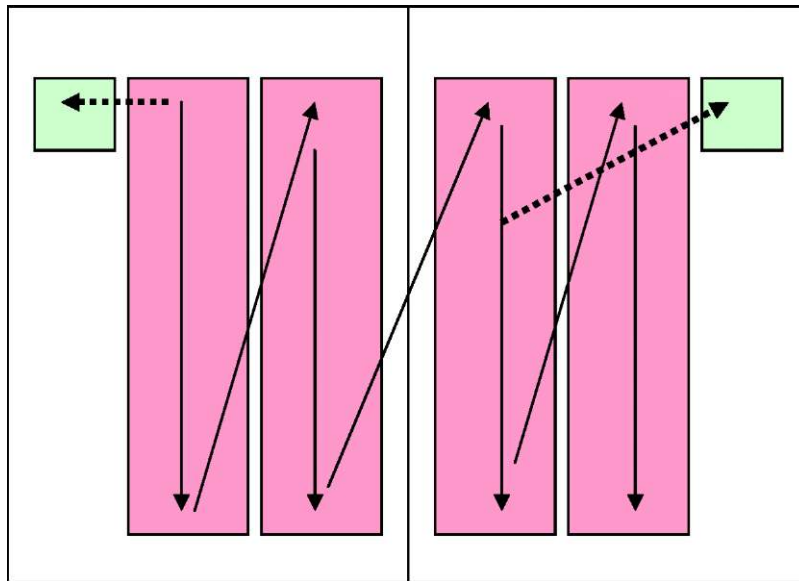
Labels preserved when XML content is imported using "Do not import contents of whitespace-only elements" option

This option is really useful when combined with another XML Import Option, **Clone repeating text elements**. When these are used together in an InDesign document that contains placeholder elements, the incoming XML elements that match the placeholder tags come into the text frame, along with the labels you created between the XML elements. So, in the InDesign layout, you get the labels, such as Major: and Note: within every repeating block, without having to type them over and over again.

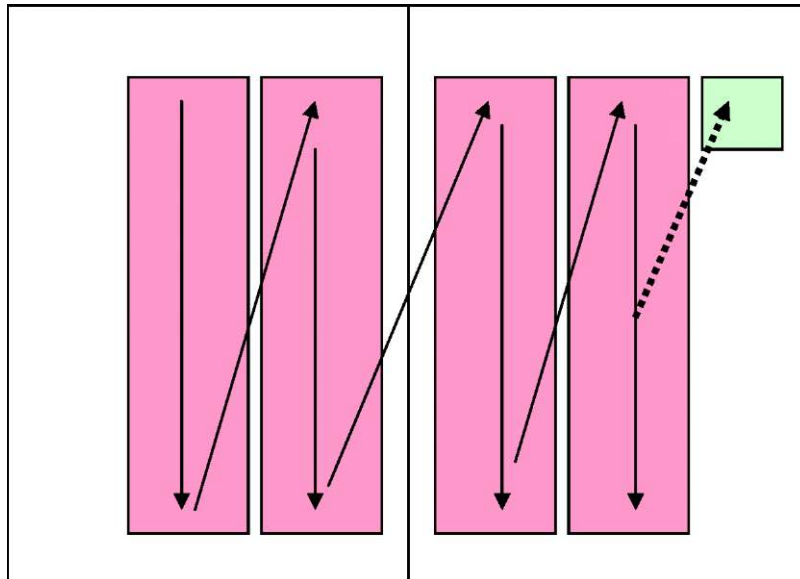
Note: to get the placeholder and cloning features to work properly, you make a template (.indt) file of the placeholder content. Then use that template to create a new InDesign file and import the XML using **Clone repeating text elements**.

Deleting non-matching structure, text, and layout components

This **XML Import Option** provides a way to "clean up" your layouts during import. It is very handy for getting unused layout objects off your pages. Most likely, you would use it when working with multiple pages that have the same layouts, as in this example, which has two columns of the major text flow (pink areas), and text frames on the outside edge of the page that are tagged with one type of XML element as a heading for the page (green areas). When the XML is imported using **Delete elements, frames and content that do not match imported XML**, the headings are placed in the text frames tagged as headings, and they do not appear in the main text flow where they occur in the document order of the XML elements. If elements flow through the main text flow for an entire page without a heading element that matches the tagged text frame at the out page edge, the heading text frame disappears from the layout.



This page spread has two tagged text frames for headings in the outside columns. When heading elements in the incoming XML match the tag of the text frame, the text frames will be used. The first heading is placed in the text box on the left-hand page, while the second heading element is placed on the right-hand page. (The original location of the heading within the XML document and the final location after import are indicated by dotted line arrows.)



*This page spread has only one matching heading element, so the tagged text frame that is not needed was removed from the left-hand page of the spread when **Delete elements, frames and content that do not match imported XML** was used.*

This option is very helpful for some types of layouts, but you should be aware that once unneeded text frames are removed from the layout, re-importing the XML file or updating its link will not create any new text frames on pages where they have been removed.

^ Rule of thumb: You should be cautious with any option that removes objects from your layout programmatically. Be sure to save a copy of your InDesign file before you import XML using the **Delete elements, frames and content that do not match imported XML** option.

Importing Images

InDesign can import XML image information that will place an image in the layout. The process is described in the Adobe publications, and there are tutorials on <http://www.adobe.com/products/indesign/crossmedia.html#xmlID>. The approach is different enough from the work that I have been doing with XML and InDesign (dealing with article layout rather than catalog content) that I recommend that you try the tutorial for the experience. (In my course catalog, we only imported text, as the images were all added later by the publishing professional who did the final layout.)

If you look at the 02_a.indd and 02_b.indd files in the tutorial zip, http://www.adobe.com/products/indesign/ID_XML_Tutorial.zip, you will see that there are many text and graphics frames tagged in the sample layouts. These layouts are pretty interesting, and I think you'll be excited by the possibilities.

These are the main points:

- If the image will not be inline, InDesign expects an image placeholder (a graphics frame) to be tagged as an element with a *required* attribute named "href." The tagged graphics frame lets you place the image in a location that is not in the main text flow.
- The value of the href attribute is the path to the local file, such as **file:///C:/adobetut/pictures/ID_XML_Tutorial/Links/01 f.psd**, where the linked image file resides. (Note the use of **lie:/** in the path.)
- In the XML file, these will be simple element structures in the form
`<Image href=11file:///path/imagefilename1.psd"/>`,
`<Image href=11file:///path/imagefilename2.tif"/>`,
`<Image href="file:///path/imagefilename3.ai"/>`, etc.
 and multiple images can be grouped inside a wrapper such as a `<PictureGroup>` element.
- Captions are not created inside the graphics frame, but are created in separate text frames.

The only challenges I see are that you need to know the path and filename of each of the images that you want to import, so, in essence, you need to have a pretty complete layout concept ahead of importing the XML.

Images come in at their actual size, but you can shrink them to fit your graphics frame (see the Help files for information on shrinking to fit).

See also the section **Tagging Images as XML in InDesign** for the tagging steps if you don't want to do the brief tutorial from Adobe mentioned above.

Note: Despite what it says in the InDesign documentation, it didn't seem necessary to create an `<Image>` element for importing an inline graphic. I think the mention of tagging a graphic with an `<Image>` tag is to get the automatic href attribute when you are creating XML to *export* from InDesign.

Note: You can create the required href attributes from differently named XML structures using the **Apply XSLT** option in the **XML Import Options** dialog. You would have to write an XSL template to transform the source XML element into the structure that InDesign requires. See the **Advanced Topics** [chapter for information](#).

Inline image imports

The XML that you use to import an image inline is very simple in structure:

```
<BannerImage href="file:///images/slogan_m.gif"/>
```

(In this case, the GIF file was in an /images folder below the folder where my InDesign document resided.)

Inline images stay where they occur in the text flow. You can't easily get them into a separate graphics frame once you have dragged the main XML file into a text frame.

Tagging XML in InDesign

The Case for Tagging Content: Why You Need XML

What makes sense for you to import or export as XML should be driven by a business need. The information in the XML should be valuable enough to justify the time and effort to mark up the content as XML and export it, or to create a template to import it.

Business functions, such as sales, marketing, manufacturing, shipping, etc., rely upon documents of various kinds to transmit information. If you look at documents, you can usually discern the function the document serves and who needs to use the document. If you look closely at documents that seem fairly "free form," such as marketing collateral, you can ferret out tidbits of discrete information within the text and images.

Try examining a piece of marketing literature, and seeing what it really contains. Typically, in the small print are legal disclaimers, copyrights, trademark notices, and the like. Does the business have a way to control the wording and usage of these important pieces of content in all of their printed materials and web pages? If these are not controlled, sometimes expensive lawsuits result from leaving them off, letting them become outdated, or providing erroneous information.

Now, look at the typical contact information and branding—company logos, slogans, addresses, phone numbers, web, email and street addresses. These also can differ from time to time, and from use to use, if they are retyped at the time that a marketing piece is created. Without a single source for these tidbits, there is the opportunity to omit or misspell every time a new content version is created.

If these small bits of scattered information don't seem like they amount to much, imagine that the company has decided on a complete rebranding, or has just been acquired. What will the effort be to locate and change every trademark, logo, address, and slogan on every type of marketing literature, support document, user guide, and other company documents?

What if the company needs everything to be provided in 10 languages for the European market—how do you let the translators know the difference between company trade names, software commands, or product names and more general text in the documents? Someone will have to provide lists of words and phrases marked as "do not translate." If these are already marked as XML, it is simple to indicate that <tradename>, <command>, or <prodname> element content should not be translated.

Tagging existing document content as XML provides the means to extract it in meaningful ways for use in business processes.

Tagging for Import

The most basic way to create XML imports is to create placeholders in an InDesign template. The key issues are: understanding where you want the XML to come in to the InDesign layout, how it should look, and what the structure of the incoming XML will be. For more information, see **Importing XML**.

In business terms, you are creating an output in a nicely formatted, printable document form to meet a business need. You hand out a business card to assist in following up a sales lead, or you give someone a handy quick start guide for a newly purchased product. You provide a set of product specifications so that someone can decide if a product meets her needs. All of these are good reasons to use InDesign to deliver an aesthetically pleasing business document. It makes the most sense when you can take the leap to seeing InDesign as one delivery mechanism among many options (web page, phone solicitation, multimedia presentation) that can connect you to customers or suppliers or partners. Using the same content across various delivery formats is leveraging the content creation process to streamline processes and reduce errors.

Tagging for Iterative XML Development

Because InDesign does not support schemas (only DTDs), you can't easily control whether people are going to put text, numbers, or dates in a particular manner in your XML elements. In database terms, there isn't anything in InDesign that will enforce "data typing" in the XML that will be valid for doing calculations or other operations. If you accept this limitation, and generally view InDesign as a generator of text content, you will be fine with a database that expects text content in data fields.

If you really need to constrain the contents of an XML element to be a numeric value, a date/time or other data type than text, you will probably not be happy with the InDesign's XML limitations in this regard. Within InDesign itself, the values of XML elements will be treated as text only. If someone types numbers, they are not truly numbers (integer or float) as far as any XML export is concerned.

XSLT would let you perform "casting" from the text values in elements to some other data type. As a post-export process, you could change text numbers into true numeric values, for example. Refer to O'Reilly's *XSLT Cookbook* for details.

Working without an initial DTD

For iterative development without a DTD, you look at the end result that you want, and the type of content that you are creating in InDesign, and design an output that will flow into the next process as simply as possible. This type of process works best for simple content that can be tagged in InDesign and mapped to a fairly shallow set of XML elements in the output.

Looking Forward: InDesign as an XML "Skin"

If it has not already occurred to you, consider this: if you have a number of XML documents, all based on the same tags, you can make them look completely different just by using a different style mapping and page layout.

For example, in template A.indt, you have 3 columns, justified text, with Caslon Old Style as the base font. In template B.indt, you have a single wide column of left-aligned text with a narrow side bar, with Helvetica as the base font. In each template, you have mapped the XML tags to paragraph and character styles of the same name (but different definitions) and applied tags to text flows. By importing the XML of the same tags structure into the different InDesign templates, you will get completely different-looking documents.

The power of this technique is only beginning to be appreciated. It is held back by the fact that there is so little standardization of XML that people use in InDesign. I expect that the next development will be the introduction of XML standard tag sets (DTDs) for publications that are rich enough to describe information usefully, but not so deep that they are difficult to use. Eventually, there will be methods to create, tag and flow XML content into InDesign that are seamless compared to the state of the art when CS5 was released.

If you want to explore this concept now, it is easy to try out.

1. First create an IDD template with styles, column layout, etc. that you like.
2. Use an XML file as the basis to create placeholder content with the tags you want to use. Then map the tags to styles in the template.
3. Save a copy of the .indt file with a new name.
4. Open it and redefine the styles, change the column layout, etc. to make a different looking design.
5. Import XML into an IDD file based on the first template. It should format itself with your styles.
6. Import the same XML, or other XML based on the same tags, into an IDD file based on your second template. It should format itself with your styles.

If you maintain the same set of XML tags and IDD styles and the mappings between them, you can create as many different document looks as you care to design. As a designer, you can look at this in a similar manner to creating HTML skins for web sites. You can commoditize the template design if you can get people to use the same XML tags for different content (or convert XHTML to XML using XSLT, as described in **Upcasting from HTML to XML for InDesign Import** in this book.)

Exporting XML

Marking Up (Tagging) Existing Content for XML Export

The "tagging" process is generally quite straightforward. Select objects or text, and then click a tag name in the Tags window's list of tags (assuming that you have loaded tags). See the InDesign **Help** section **Tagging content to export to XML** for more information. For tables and images, I have provided more details in this Short Cut.

The Special Case of InDesign Tables (Namespaced XML)

A surprising number of layout people have never tried the table features of InDesign, preferring to group text frames to make tables. But because InDesign CS offers powerful table design and production, we will make good use of them for our XML processes.

We'll work through some table exercises. First, to understand how InDesign represents a table as XML, we will start by saving a tagged table as XML.

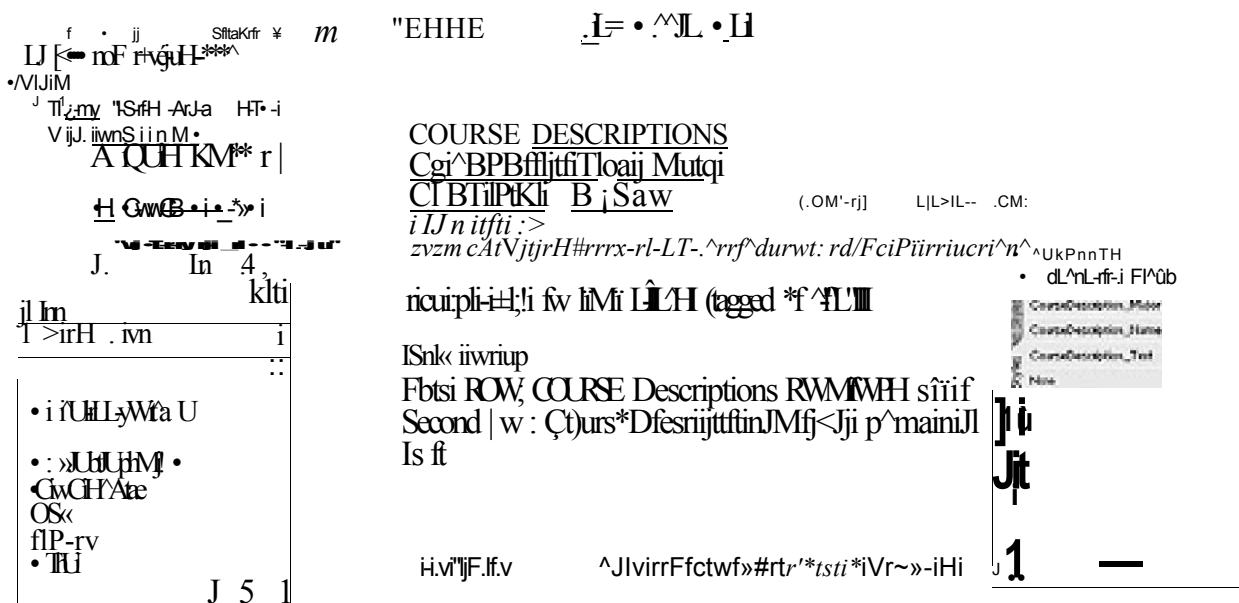
1. Select **Table>Insert Table**. The **Insert Table** dialog opens. Create this table as two columns, with one heading row, and three normal rows.
2. Now, click in the top left edge of the first row to select the row, then select **Table>Merge Cells** to combine the two cells into a single row.
3. Repeat the **Merge Cells** operation for the second and fourth rows of the table, leaving the third row as two separate cells.
4. Now use **Table>Select>Table** to make sure the entire table is selected, then use the **Tags** palette to tag the entire table as `<CourseDescriptions>` (the root element of the sample XML file).
5. Select the top row, type COURSE DESCRIPTIONS as the heading for the entire table. Create a new Paragraph style named TableHead1 and make the text bold and large, and apply it to the top row of the table. Do not tag the top row as an XML element.
6. In the second row, type CourseDescription Major. Apply the matching Paragraph style, then select the row, and tag it with the `<CourseDescription_Major>` XML tag name.
7. In the first cell of the third row, type CourseDescription Name. Apply the matching Paragraph style, then select the cell (not the whole row), and tag it with the `<CourseDescription_Name>` XML tag name.

8. In the second cell of the third row, type "CourseDescriptionText." Apply the matching Paragraph style, then select the cell (not the whole row), and tag it with the <CourseDescription_Text> XML tag name.
9. In the fourth row of the table, type "CourseDescriptionFootnote." Apply the matching Paragraph style, then select the row and tag it with the <CourseDescription_Footnote> XML tag name.
10. Below the table, type "Course offering subject to change. Check the web site for current offerings." Create and apply a new paragraph style named Note that is a variation of the CourseDescription Footnote Paragraph style (smaller and in a different font so that it looks distinctive).
11. Save your file containing your styled and tagged table.
12. Also use Save As Copy in the file menu to freeze a copy of this file as a snapshot to which you can return.

Examining the table

Open the **Structure** pane if it is not already open. If you watch the Structure pane while you move your cursor from place to place in the table, you will see a highlight on the element that has been used to tag each row or cell.

Notice there is a new element in the Structure pane named Cell that contains the text for your table heading that you styled as TableHead1. Notice that in the Tags palette, there are two new elements: <Table> and <Cell>.



The **Structure** pane and **Tags** palette (left), the table (center), and the **Paragraph Style** palette (right), showing that InDesign adds a new tag named Cell when untagged content is included in a tagged table

In CS3, CS4: When you use **Edit>Edit in Story Editor** to look at your file, instead of seeing the angle symbols with XML element names around the text, you will only see the element names in the left panel (the **Structure** pane). InDesign CS3/CS4 does not represent the tagged table content in the **Story Editor** view.

In CS5: When you use **Edit>Edit in Story Editor**, you will be able to see how each individual table cell is tagged with XML and what it contains.

InDesign treats tables differently from all other XML. Tables are a specific type of object within the InDesign application, different from text objects or media (image) objects. So InDesign has arbitrarily added new `<Table>` and `<Cell>` elements to your set of XML tags to accord with its internal representation of the table object.

Even more surprises await you when you export this test table as XML.

We'll perform two trials. Select the entire text frame containing the table. While it is selected, select **File>Export**, then choose XML as the file type and name the file "placeholderTable01.xml." Check the box to see the resulting XML in Internet Explorer. Leave the encoding as UTF-8 (the default). Then, click **Export**.

Now, in the **Structure** pane, click in the `<CourseDescriptions>` element, then right-click to get a context menu. Select **Export from Selected Item**, and check the box to see the resulting XML in Internet Explorer. Type a different name for this file, such as "placeholderTable02.xml." Leave the UTF-8 encoding, then click **Export**.

When you view the second file you made (placeholderTable02.xml) in Internet Explorer, you see it contains the XML elements within the `<CourseDescriptions>` element, including the `<Cell>` element that InDesign added to tag the first row.

In addition, you see a lot of XML markup that you did not explicitly apply. The top line now declares an XML namespace called "aid:" with this "xmlns" attribute (`xmlns:aid="http://ns.adobe.com/AdobeInDesign/4.0/"`) and adds it to some attributes (`aid:table`, `aid:trows`, and `aid:tcols`) of the `<CourseDescriptions>` element you used to tag the table. (For CS4 and CS5, the namespace `xmlns:aid5="http://ns.adobe.com/AdobeInDesign/5.0/"` is added to support table styles).

The root `<table>` element has a number of aid: attributes:

```
aid:table="table" aid:trows="4" aid:tcols="2"
```

The header cells have markup for another set of aid: namespace attributes:

```
aid:thead="" aid:crows="1" aid:ccols="2">
```

And the other cells also contain aid: namespace attributes, such as
 aid:table="cell" aid:crows="1" aid:ccols="2"
 and aid:ccolwidth="206.5010000000001" (some long decimal value).

A complete example looks like this:

```
<CourseDescriptions xmlns:aid="http://ns.adobe.com/AdobeInDesign/4.0/"
  aid:table="table" aid:trows="4" aid:tcols="2">
  <Cell aid:table="cell" aid:thead="1" aid:crows="1" aid:ccols="2">COURSE
    DESCRIPTIONS</Cell>
  <CourseDescription_Major aid:table="cell" aid:crows="1"
    aid:ccols="2">CourseDescriptions_Major</CourseDescription_Major>
  <CourseDescription_Name aid:table="cell" aid:crows="1" aid:ccols="1"
    aid:ccolwidth="206.5010000000001">CourseDescriptions_Name</CourseDe
    scription_Name>
  <CourseDescription_Text aid:table="cell" aid:crows="1" aid:ccols="1"
    aid:ccolwidth="332.99900000000014">CourseDescriptions_Text</CourseDe
    scription_Text>
  <CourseDescription_Footnote aid:table="cell" aid:crows="1"
    aid:ccols="2">CourseDescription_Footnote</CourseDescription_Footnote>
</CourseDescriptions>
```

Example of aid:namespace on table XML

The aid: namespace is used for many functions of InDesign "under the hood." For now, it is only important that you see that the markup appears in the XML export even though you did not explicitly add it in InDesign.

Now, open the first table XML file you exported. You will see that this exported XML file contains the automatically generated Root and Story elements and, near the bottom, the untagged text you styled as a "Note" Paragraph style.

```
<Root>
- <Story>
- <CourseDescriptions xmlns:aid="http://ns.adobe.com/AdobeInDesign/4.0/"
  aid:table="table" aid:trows="4" aid:tcols="2">
  <Cell aid:table="cell" aid:thead="1" aid:crows="1" aid:ccols="2">COURSE
    DESCRIPTIONS</Cell>
  <CourseDescription_Major aid:table="cell" aid:thead="1" aid:crows="1"
    aid:ccols="2">CourseDescriptions_Major</CourseDescription_Major>
  <CourseDescription_Name aid:table="cell" aid:crows="1" aid:ccols="1"
    aid:ccolwidth="206.5010000000001">CourseDescriptions_Name</CourseDescrip
    tion_Name>
  <CourseDescription_Text aid:table="cell" aid:crows="1" aid:ccols="1"
    aid:ccolwidth="332.99900000000014">CourseDescriptions_Text</CourseDescrip
    tion_Text>
  <CourseDescription_Footnote aid:table="cell" aid:crows="1"
    aid:ccols="2">CourseDescription_Footnote</CourseDescription_Footnote>
  </CourseDescriptions>
    Courses offerings subject to change. Please check the web site for current
    course offerings.
  </Story>
</Root>
```

Notice that while the aid: namespace was added to <CourseDescriptions> and its child elements, it does *not* appear on the <Root> or <Story> elements or the untagged line of text.

From these two examples of exported XML, we can reverse engineer what will make InDesign "understand" and format imported XML as a table. The aid: namespace informs the program that we are using its underlying layout representation. To create a table on import, we need to add the aid: namespace attributes to each XML element.

Interpreting the aid: attributes is fairly straight-forward. The element that will become the table (generally some sort of wrapper element in the XML) minimally needs `aid:table="table"` and an `aid:tcols` attribute set to the value of the number of columns in the overall table layout, such as `aid:tcols="4"`.

Note: There is no <row> in InDesign tables. The value of `aid:trows` is optional for the element with the `aid:table="table"` attribute. If set, it may truncate imported XML at the specified number of rows. If left off, the rest of the imported content will determine the total number of rows in the table.

The values of the attributes `aid:ccols` and `aid:crows` govern the column and row spans, respectively, of an element with an attribute `aid:table="cell"`. So, if on the table element the value is set with `aid:tcols="2"` and the cell-level element has the `aid:ccols="2"`, then the cell-level element will span two columns, the width of the table. If `aid:crows` value is greater than 1, the cell will span the specified number of rows in the table.

Here is a little reference to help you keep this sorted out:

Namespaced attribute name	Purpose
<code>aid:table</code>	Defines any table-related element
<code>aid:table="table"</code>	Defines the table element itself
<code>aid:tcols="N"</code>	Defines N number of columns in the table
<code>aid:trows="N"</code>	Defines N number of rows in the table. Note: You do not have to supply a value for this attribute, and you should omit it when importing XML to avoid having the XML truncated when it reaches the "N" value of <code>aid:trows</code> .
<code>aid:table="cell"</code>	Defines a cell within a table
<code>aid:ccols="N"</code>	Defines N number of columns that the cell will span.

Namespaced attribute name	Purpose
<code>aid:crows="N"</code>	Defines N number of rows that the cell will span.
<code>aid:thead=""</code>	Makes the cell a header, which enables it to be reused automatically at the top of every text frame for tables that flow from one text frame to another. Note: this is always an attribute with an empty value "" because InDesign assumes that if it appears, it should be applied, so you don't have to type a "true" or false" for aid:thead.
<code>aid:tfooter=""</code>	Makes the cell a footer, which enables it to be repeated at the end of every text frame for tables that flow from one text frame to another. Note: this is always an attribute with an empty value "" because InDesign assumes that if it appears, it should be applied, so you don't have to type a "true" or false" for aid:tfooter.
<code>aid:colwidth="NNN.nnnnn"</code>	Width of column (in points not pixels, approx. 72 = 1 inch)
<code>aid5:tablestyle</code>	Starting with CS4, the ability to add a named table style to the table object. You need to have created a placeholder table with the named style in your InDesign template, for the style to be applied automatically on importing XML into InDesign.

Table of InDesign's aid:table namespace elements

You can find more information on aid: namespaces for tables on pages 20-21 of **Adobe InDesign CS3 and XML: A Technical Reference**

http://www.adobe.com/products/indesign/scripting/pdfs/indesign_and_xml_technical_reference.pdf (or search online for aid: or aid5: namespace)

Tables are tricky to work with, and if the namespace is confusing, the easiest way to create them is with the placeholder technique.

Tagging Images as XML in InDesign

Images are another special case in InDesign. Because a linked image is actually a separate file, InDesign needs a pointer to the linked image in a graphics frame in the document. So, you create a graphics frame, place a linked image into the frame (see the InDesign **Help** if you don't know how to do this), and tag it as an `<Image>` element. Then, you will need an attribute of the `<Image>` element named "href" and the path to the file folder and image file as its value, i.e.

```
<Image href="path/to/folder/myfineimage.psd">.
```

Adobe has kindly provided some assistance; according to the **Help**, "When you tag a graphics frame [with the Image tag], a reference to the graphic's location (on disk) is placed in the exported XML file." (If you use a different element name than `<Image>` to tag an image, the element is required to have the href attribute with the file path to the image file, the same as an `<Image>` element.)

Image Options in the Export XML Dialog

When you export XML that includes the tagged, linked images, InDesign provides a tab in the **Export XML** dialog with some choices. The **Image Options** are geared mostly to creating web-friendly image output. But, you can simply copy the original images to a folder by checking the first checkbox, **Original Images**, below **Copy to Images SubFolder**.

For the other two checkboxes, InDesign provides the same features: **Image Conversion (Automatic**, which decides the file format based on image content, or **GIF** or **JPEG** for all images), and settings for color palette and image quality, etc.

Expert XML

General Images

[Image Options]

Copy to Images Sub-Folder:

☐ Optimize Images

☐ Formatted Images

☐ Formatted Images

Image Conversion: Automatic

a

Note: Formatted Images have drop shadows or scaling to fit, etc. which can be maintained on the optimized images.

GIF Options

Palette: Adaptive (no dither)

☐ interlace

JPEG Options

Triage Quality: Low

LZW Method: ftselne

Cancel

The Export XML dialog's Images tab, which provides optimization for web images

If you choose to optimize images (formatted or not), the XML output includes special attributes that point to the subfolder (automatically named /images) and also to the location of the original image. In this example,

```
<Image href_fmt="images/01_f_fmt.jpg"
href="file:///C:/adobeTutorial/ID_XML_Tutorial/Links/01_f.psd"/>
```

the href_fmt attribute contains the relative path to the optimized image from the current InDesign document. The href attribute points to a local file folder location of the original image that was placed in the document.

If you choose to copy original images to the subfolder, then your .psd, .ai, or whatever format image files would simply be duplicated in the /images folder.

Note: You can use XSLT to change InDesign's <Image> elements into a different element, such as the HTML and just copy the href onto it. Or you can write a different href value (path to a server folder). If you want to do this, I recommend testing it first as a post-export transformation, rather than as **Apply XSLT on Export** (which is more likely to crash).

Validating XML in InDesign

A foolish consistency is the hobgoblin of little minds...

—Ralph Waldo Emerson

Why validate?

Validation is the process of assuring that the XML you create conforms to the rules of your structure, whether governed by a DTD, schema, or database structure. While "a foolish consistency" in philosophy may not be praiseworthy, a wise consistency in content creation may save you a lot of later effort.

When you validate your XML content, the assumption is that any other system using the same structural rules can use the XML you create. For solo content creators who are only making XML and storing it for their own uses, validation may not be necessary. But for anyone who is working with XML with the intent of passing along the content they create to other applications or processes, making sure that the XML is valid will be imperative.

How to Validate XML in InDesign

Adobe has given you the ability to make mistakes when creating XML content, then go back and fix them by checking the content's validity against a DTD. If you need really robust validation while authoring XML content, InDesign may not be the right choice for your publishing needs. Adobe FrameMaker and other XML publishing applications that offer a guided content creation process (preventing you from inserting invalid structure in your document) will serve you better.

Consider InDesign as a tool for which validation should be done before importing XML (if possible), or frequently while creating XML structure within an InDesign document.

To use InDesign's validation features, you must first load a DTD. If your organization is using XML elsewhere, you may have been provided a DTD that you can use for creating XML in InDesign. If not, you may want to consider what type of DTDs already in existence would serve your needs.

DITA (for topic-based content that is suitable to repurpose as online help), NewsML (for news articles), and other useful rulesets for technical publishing, catalogs, theses, and scientific papers are available if you search online. I would advise the use of the simplest DTD you can find that will provide the minimum structure you need. Most DTDs have low-level structures (titles, subheads, tables, lists, links, images, and generic paragraphs) that will be very similar to HTML tags. More complex DTDs also provide "semantic" tags that help you understand the meaning of a piece of content, such as bylines, author names, product

specifications, units of measure, mathematical equations, chemical structures, etc. Use the simplest DTD that will meet your needs, if you don't already have one.

You should work locally with a copy of the DTD, rather than pointing to a DTD on a remote server on the Internet. Save the DTD on your own computer in the folder with the InDesign documents, or in a common server that a group of content creators can access. Just remember two things: you should use the markup in the way that the DTD was intended (not by guessing, but by having documentation that explains how it is to be used), and you should *never* alter a DTD yourself if it is being used by a group of people.

Loading a DTD and getting the correct root element

To add a DTD to your InDesign document, simply open the **Structure** pane, click on the upper right corner arrow icon to get the **Structure menu**, then select **Load DTD**. Browse to where the DTD is located and select it. If you do this in an empty InDesign document, you will automatically get a root element tagged with the default <Root> element that InDesign provides. InDesign will load all of the tags from the DTD in alphabetical order into the **Tags** palette when you load the DTD. To apply the root element from your own XML tags, open the **Tags** palette, select the default <Root> element, and retag it with the real root element of your DTD.

Authoring with a DTD

After you have loaded the DTD and applied the correct tag to the root element, you can start authoring and validating as you go. Here is an example DTD:

```
<?xml version="1.0" encoding="UTF-8"?>
<!--DTD generated by XML Spy v4.3 U (http://www.xmlspy.com)-->
<!ELEMENT CourseDescriptions (CourseDescription_Major | CourseDescription)+>
<!ELEMENT CourseDescription_Major (#PCDATA)>
<!ELEMENT CourseDescription (CourseDescription_Name | CourseDescription_Text |
CourseDescription_Footnote)+>
<!ELEMENT CourseDescription_Name (#PCDATA)>
<!ELEMENT CourseDescription_Text (#PCDATA)>
<!ELEMENT CourseDescription_Footnote (#PCDATA)>
<!--ATTLIST CourseDescription_Footnote
    type CDATA #REQUIRED-->
```

A simple DTD for course descriptions

This DTD states that I have a root element <CourseDescriptions> that can contain multiple <CourseDescription_Major> and <CourseDescription> elements. There aren't any structures inside a <CourseDescription_Major>, only text (PCDATA).

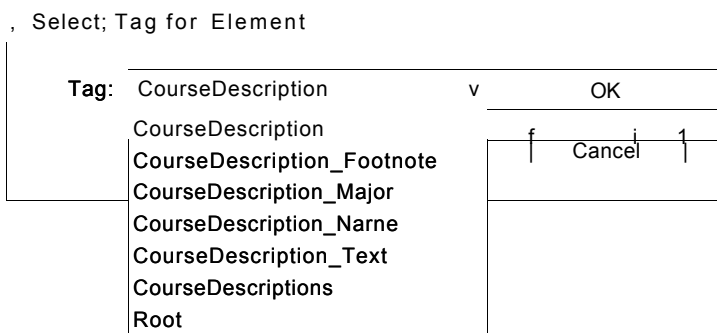
Within a `<CourseDescription>`, there are `<CourseDescription_Name>`, `<CourseDescription_Text>` and `<CourseDescription_Footnote>` elements. These elements don't contain any other element structures.

`<CourseDescription_Footnote>` has a required attribute of type (used mainly to indicate when there is a prerequisite for the course mentioned in the footnote).

(This is an extremely simple DTD. I will not go into detail about how to read DTDs and how to write them, but suggest that you read up on them online or in a book like *XML Elements of Style*.)

Create a text frame and tag it with the root element of your DTD by dragging the root element onto the text frame (use the Text selection tool when dragging). To see if the tag has been applied, use **View>Structure>Show Tagged Frames**.

Now, with the text cursor in the tagged text frame, select the square **New Element** icon in the top bar of the **Structure** pane. In the dialog that opens, select the next element that you need from the drop-down list (this can be very long for a complex DTD), and it will be added to the structure, indented below the root element.



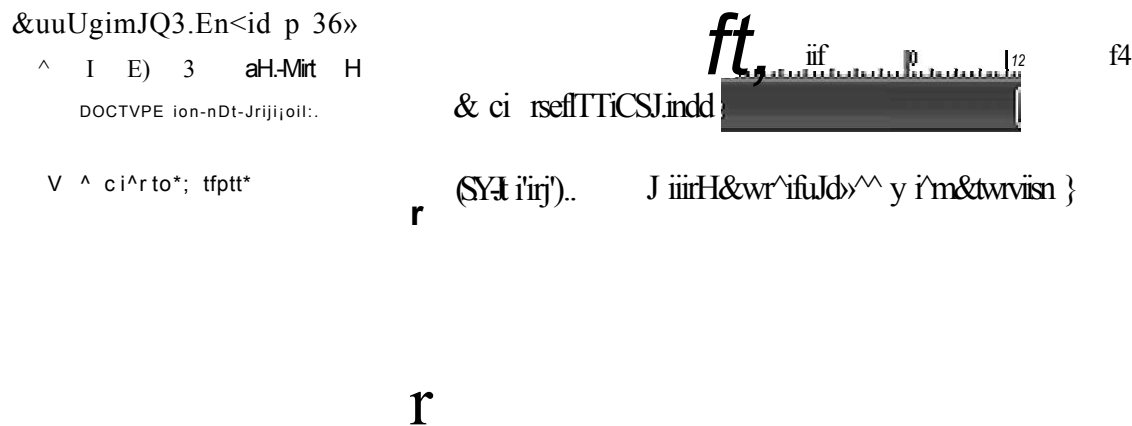
Adding an element with the Structure pane icon (by selecting a tag name from the drop-down list)

You can test if you added the first element that the DTD requires by clicking the **Validation** icon (lightning bolt in the top bar of the **Structure** pane). A subwindow appears at the bottom of the Structure pane and will display the words "no known errors" if you selected the correct element to add below your root element.

Dealing with validation problems

If you see that the element you inserted has turned red when you click the **Validation** icon, the validation window will display an error message and suggestions for how to fix the structure. If you click on the suggestion that you think will solve the validation problem, the suggested fix will be applied in the **Structure** pane. If it is correct according to the DTD, the element will no longer be red in the structure pane.

One problem with the suggestions that InDesign makes is that many of them are not helpful to the process of creating content quickly. For example, if I insert a <CourseDescription> element inside a <CourseDescription_Major> element, InDesign states that the element is not valid in that position in the structure, and suggests that I delete the element.



Err*n
Thk *J)nvti* if ft» Vj U A) lhri fifJKrt.
FCF-bk ~~FKK~~
jgaaaSeimaaafirt; T-

InDesign's suggested solution for an element in the wrong place

While this action will fix my problem, it is not the best solution. The DTD requires that the <CourseDescription> element be at the same level of indentation as the <CourseDescription_Major> (a "sibling" element, rather than a "child"). It would be better for me to move the <CourseDescription> element out of and below the <CourseDescription_Major> element, as my intention was to add a <CourseDescription> to the structure. To move an element, select it in the **Structure** pane, and drag it to the new location (the location where it will be dropped into the structure is indicated by a bold line).

If I validate the structure now, the <CourseDescription> element turns red and multiple suggestions are displayed in the validation window. InDesign recognizes that the <CourseDescription> element must contain other elements (for the name, descriptive text and footnotes) to be valid.

Note: Use **Edit>View in Story Editor** to see the element tags more clearly when you are not typing text within the elements. You can also add line breaks and rearrange (drag) elements in **Story Editor View**.

& coiirseDTDCS3.indd @ 42%

i # S f Stmcture •
 DOCTYPE CoureeDescriptions
 V [jil_ CoureeDescriptions
 [jil_ CourseDescription
 [M1 CourseDescription_Major

CourseDescription**Error:**

A required element is missing.

Possible Fines:

Insert 'CoureeDescription Name' element

Insert 'CoureeDescription Text' element

Insert 'CoureeDescription Footnote' element

X**•A****Why authoring XML with a DTD is problematic in InDesign**

1. It is easy to accidentally retag the root element in the **Structure** pane with an invalid element name.
2. Multiple suggestions may display in the validation window, so to select the correct one, you must understand the DTD in depth.
3. InDesign cannot move elements to the correct locations in a structure for you, so it will suggest that you delete elements that are inserted in the wrong location. In practice, you usually will want to drag the element to a different location.
4. The **Structure** pane drag-and-drop action can be tricky to do correctly on long documents.

Multiple suggested fixes in the validation window (lower left)

I can click each of the suggestions to insert the missing elements. InDesign will add them to the Structure in the order that I click the suggestions. Or, I can use the **Add Element** icon to insert the three subelements inside the <CourseDescription> element. I just select <CourseDescription> in the Structure pane, and then add the other elements one by one in the order that I want them to appear.

When I validate, this time the <CourseDescription_Major> and <CourseDescription> are not red, so it has been fixed. However, the <CourseDescription_Footnote> I added is red, and the validation window suggests that I add the required "type" attribute.

© courseDTDCS3.indd @ 72%

i	•	E	Structure	•
[^1	DOCTYPE	CoureeDescriptions		A
^	CourseDescriptions			
^	[M1	CoureeDescription		
	[M1	CoureeDescription_Name		
	[M1	CourseDescription_Tej(t		
	JM1	CoureeDescription_Footnote		
	[M1	CoureeDescriptionJMajor		
				V
Kill		N	>	
	CourseDescription_Footnote			X
	Error:			
	A required attribute is missing.			
	Possible Fines:			
	Add attribute 'tvpe'			

A validation error and suggestion for a missing attribute

When I click the suggestion, an empty attribute name "type" is added to the `<CourseDescription_Footnote>`. When I click the **Validation** icon, the structure is now valid, even though I have not supplied any value for the "type" attribute "

If I double-click the "type" attribute, the **Add Attribute** dialog opens, and I can type in the value for this attribute, such as "prerequisite." I need to know what the choices are for the value of the attribute, if they are constrained, or whether I can put in any value I want (in this case, I can). So again, it is important to understand how your DTD works in detail to author valid XML in InDesign.

Occurrence and sequences of elements

Many DTDs limit the occurrence and sequence of elements at different places in the structure they define. Only one element of a given kind, or a sequence of elements in a certain order, may determine how you author XML in InDesign. Such constraints add to the complexity of the validation process.

Note: Validate early and often when working with XML and DTDs for the first time. Once you have created a set of valid structures, you can write a procedure to follow so that your next XML document authoring session will be less difficult.

In the case of my DTD, the `<CourseDescription_Major>` and `<CourseDescription>` element can occur multiple times, and the sequence is not constrained. So, I can have a `<CourseDescription>` element before a `<CourseDescription_Major>`

element, even though that is probably not what I really want. But I will resist the temptation to "improve" the DTD by adding more constraints to the structure model. Logically, any `<CourseDescription>` might be a child of a `<CourseDescription_Major>`, but there might be courses that are not associated with a major. The database model that was the origin of the elements for this DTD is not constrained in this regard, and I want to maintain synchronization with that database.

Note: Schemas (not DTDs) based on database tables can be generated from many database applications. InDesign does not support schemas, so you will need to use another XML tool to convert the schema to a DTD to use in InDesign.

Validating outside of InDesign

As a reality check when working with a DTD, even when InDesign reports "no known errors," you can double-check the XML's validity. Export the XML with the choice to **Include DTD declaration** checked in the **Export XML Options** dialog.

Export XML

General images

Options

☐ include DTD Declaration

☐ View XML Using: »xplore-SBB

☐ Export From Selected Elements

• Export Untagged Tables as CALS XML

☐ Remap Break, Whitespace, and Special Characters

• Apply XSLT: [\[Use Script In XML\]](#)

U

Encoding: UTF-8

Export the DTD declaration in the XML output for external validation

Now, the XML can be tested in another XML application that has validation capabilities. In XML Spy, the exported file opens without any warnings if the file is valid. Here is what my valid sample (empty) XML elements look like:

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<!DOCTYPE CourseDescriptions SYSTEM
"C:\Textenergy\oreillybooks\shortCut\coursedescriptions.dtd">
<CourseDescriptions>
  <CourseDescription>
    <CourseDescription_Name/>
    <CourseDescription_Text/>
```

```

    <CourseDescription_Footnote type="whatever"/>
  </CourseDescription>
  <CourseDescription_Major/>
</CourseDescriptions>

```

Valid XML with DTD declaration exported from InDesign

There are many other things that you should know about working with DTDs, schemas, databases, and creating valid XML. Consider acquiring a book or researching these subjects online if you plan to become proficient in XML.

Duplicating structure to build XML

Once you have a block of elements that is valid, you can duplicate the structure to build your XML document more rapidly. Select an element structure that you want to copy in the **Structure** pane. Then, use **Copy** to put it on the clipboard. Select the copied element structure's *parent* element and **Paste**. The duplicate structure will be added to the end of the current structure of the parent element.

Note: Paste operations add content to the end of the parent structure. In InDesign, you can't Paste an element structure between other elements that are siblings. If you want an element or structure to be placed in between existing sibling structures, you have to drag the new XML content after you paste it.

Duplicating is much faster than tagging plain text paragraphs with XML elements after you type the text. When you have created blocks of duplicate structures with **Copy/Paste**, you can switch to the **Story Editor**, and start typing text into the elements.

Note: Don't forget to switch back to the **Layout** view to see how your line breaks, spaces and tabs look on the page. While the **Story Editor** is great for seeing the start and end tags of your elements, it is easy to forget to insert breaks and have XML elements that run together on the page.

Cleaning up imported XML content

You should run the validation on any XML that you import into InDesign for which you have a DTD. With a large XML file, this can be problematic, because you will have a large number of suggestions in the validation window to work through. If possible, validate the XML *before* importing it into InDesign.

Fast and Light Credo: Develop Now, Validate Later

If there is no DTD available for you, you can create one from sample XML yourself. Start by making a list of the different information bits that will make up your content structure. For example, suppose that I am trying to get some personnel information into my course catalog. On looking at the text in the current unstructured document, I see that there are people who have administrative positions, people who are on various boards, people who have won awards, people who are on staff, and various types of teachers on the faculty. All of these people are employees except some of the board members. All of the faculty have their degrees, degree-granting institutions, and any special awards listed after their names. In another part of the catalog, these people are listed with phone numbers.

I can choose `<personnel>` as my root element (while knowing that a few people are technically not employees). Then I can have a `<person>` with `<name>` elements patterned on a standard model like `<lastname>` `<firstname>` `<mi>` and `<honorific>`. Each person can also have contact information (`<phone>`, `<email>`, etc.) and `<degrees>` containing `<degree>` and `<institution>` elements.

With this as a starting point, I can model the XML directly in InDesign. I work in much the same way that I did when I was using a DTD, except I don't do any validating as I go. Once I have a good example set of XML elements, I export the XML. Then with another XML tool, such as XML Spy or Oxygen, I generate a DTD based on the sample.

Once I have an initial DTD, I load it into my InDesign document that contains the sample XML content and validate the XML with it. Assuming that it validates, I continue to make XML to match the structure model of my DTD.

Iterating the information structure and DTD

At some point in my development, someone may ask for a new element in the XML structure. Perhaps they want to add the teacher's department to each person who is a teacher, and an office location to each staff and administrative person. I can add this new element structure based on some database fields used for the college's online directory, or by looking at a printed directory to see what headings and information are in the directory that describe the locations of people on campus (like the building name and/or number, the floor, room or suite, etc.).

I now delete the current DTD from the **Structure** pane in InDesign because I don't want a lot of validation errors with my new structure. I add the `<location>` elements to the contact information for each person in a three-phase process:

First, I build a placeholder for the `<location>` structure with all the new elements I want in the first `<person>` element of my XML content. I export the XML again so

that the new <location> structure is included in the sample XML that I use for DTD development.

Second, I regenerate a DTD with my other XML tool, then load it into my InDesign document, and validate the XML that contains my new structure. If I have made the new <location> structure *required*, every <person> element that is missing the <location> structure will now generate an error in the validation window.

Third, I repair the XML so that it will validate with the revised DTD. I can duplicate the first <location> structure by selecting it in the **Structure** pane and doing a copy/paste operation to add it to the other <person> element structures. Then I can validate again and continue fixing the structure until I see the "no known errors" message in the validation window.

Note: If you know how to work with XSLT, this would be faster than manually adding elements with a copy/paste operation. You can write a template that adds new XML structures to your existing XML file. Consult books such as *XSLT Cookbook* or online references on XSLT for more information on adding structures or modifying existing ones.

The process of creating the new structure, generating a revised DTD, and validating can be repeated as I make changes to the structure. At each iteration, I can demonstrate the XML to stakeholders to see if it meets their needs. Once everyone has agreed to a structure, I can annotate the final DTD with comments so that everyone knows what each element is and how many times it can occur and in what sequence. This documentation then gets distributed to anyone else who needs to create XML or use the XML that is being created.

Despite all your best efforts to document how the DTD should be used, and to train authors, someone will always create invalid XML content in InDesign. It simply does not provide enough XML authoring support in its current incarnation. If the content authors are struggling, you might investigate other XML authoring tools that help authors use a DTD. Adobe makes another product, FrameMaker, which provides guided authoring for XML based on a DTD. FrameMaker costs about the same as InDesign, but requires more expert assistance to set up for XML import and export than does InDesign. Or, if you search online, you will find a number of XML editors and authoring tools (freeware, shareware or commercial) that may meet your authoring needs and budget.

If you use a different XML authoring tool than FrameMaker or InDesign itself, you can still import the XML you create into InDesign to make a visually pleasing document. In this regard, InDesign provides sufficient functionality to be very useful for XML publishing. Validate the XML before you import it. Then use the techniques described in this Short Cut to map to Paragraph and Character styles.

What InDesign Cannot Do (or Do Well) with XML

The 1:1 Import Conundrum

It's fairly natural to expect that you could use one piece of XML data in multiple places in an InDesign layout—but that's not at all the way that InDesign works. Once you've imported XML, there is a one-to-one correspondence between the elements in the Structure view and their expression in the layout. If you want an element to appear multiple times, you've got to duplicate the element for each appearance on a document page. (Obviously, you can get around this in some cases by placing the XML element on a master page.)

— Olav Martin Kvern and David Blatner, *Real World Adobe InDesign CS4*

As the quote above states (and this is still true for InDesign versions up to CS5), the expectation is that you import one XML file to fill one content area (text flow) in your InDesign document. This is contradictory to the spirit of XML, which is all about reuse of content in multiple documents and in multiple ways. For example, you might want a standard warning or copyright or other block of content to appear in many places in a single document or a set of documents collected as a book. However, from the **Structure** pane, you cannot drag the same piece of structure into multiple locations in an InDesign document. If you drag an element into the layout a second time, InDesign will remove it from its first location in the layout.

Bad Characters

Note regarding CS2 only: InDesign CS2 XML export controls are more limited than those in InDesign CS3 and later. CS2 does not have the **Remap Break, Whitespace and Special Characters** option. As a consequence, the XML that you generate from InDesign CS2 may contain characters used in publishing applications that are problematic in XML processing. Chief of these are the characters that make paragraph and manual line breaks in the text layout. XML doesn't use these types of characters, and depending on the processes you run after exporting XML from InDesign, you may have to clean up the XML to remove these types of characters.

•Course offerings subject to change. Please check the web site for current course offerings. •

Unwanted characters (square) in XML exported from InDesign CS2

Some typographic controls may generate characters even in CS3-5 that are not XML-compatible. Adobe warns about this in the **Help** section about exporting XML: "Not all characters are supported in XML (such as the Automatic Page

Number character). InDesign warns you if it cannot include a character in the XML file."

In all versions of InDesign CS: Related to the "bad characters" export problem above is the issue of imported XML that might contain tabs, spaces, and line breaks. Often this is seen in applications that "pretty print" XML files with indents and coloring to make them easier to read. For example, I use XML Spy by Altova and SynchroSoft Oxygen Editor. When the pretty-printed XML created in XML Spy is used for import, it creates unwanted effects in the layout.

To get a clean import, it is sometimes necessary to edit the text in a text editor to remove the tabs and spaces, play with the import dialog whitespace controls (**Do not import contents of whitespace only elements** in CS3 and later), or to run an XSL transformation to remove line endings and tabs from the XML before importing it into InDesign.

Inscrutable Errors, Messages, and Crashes

The Devilish DTD suggestions

Are you missing a required attribute? Have you forgotten to put a required element in your structure? The validation window at the bottom of the structure pane will tell you the sad story of your incompetence with the DTD, but the suggestions it offers won't always tell you enough about fixing the problem—see the section on **Validating XML in InDesign** for more information.

Exporting from the element with the included DTD will not be valid

Several times when I had a DTD included in the XML that I was exporting, and checked the box to include the DTD declaration on export, I saw a message that the XML I was exporting was not going to be valid using the DTD. It seemed to me at the time that the message was bogus, as I had validated the content with the DTD before export. I opened the exported content in XML Spy to check it, and found that there was some kind of invisible (line break) character in the XML between elements. When I switched to EditPlus and looked at the same file, I saw square box characters in these places in the XML file. I had to do a search and replace on that character to get an XML file that would validate in XML Spy. This is related to the issue described in the topic **"Bad characters."**

Don't make InDesign "think" too hard on import or export with XSL

It seemed to me that the most likely times to make InDesign crash were if I tried to get too fancy with my XSLT. I am accustomed to being able to sort, filter, wrap and unwrap elements, make substring operations on text in elements, and other tricks of the XSL trade. If I used these types of functions in XSLT that I was using when importing or exporting XML with InDesign, sometimes it didn't work, and

sometimes it froze the application. See **Advanced Topics: Transforming XML with XSL**. My recommendation, if you need to do a lot of fancy manipulation of your XML, would be to use XSLT as a pre- or post-processing step external to InDesign.

InDesign CS5: XML Structure option for exporting XHTML

This aspect of exporting XML relates to CS5 only. Related content is under the section **Exporting XHTML when XML is in your InDesign file**.

When you tag content in InDesign, the order of the elements can be seen in the **Structure** pane. If you first tag a heading as <title>, then the author's name as <byline>, then you jump across the page spread and tag a text frame as <sidebar>, then you return to the main story and tag it and its paragraphs with various XML tags, that will be the XML order of the content. In other words, you do not have to be in a top-to-bottom and left-to-right sequence of tagging the content with XML.

In InDesign, if you check "Base On Page Layout" when exporting XHTML, the sequence of the content is determined by the X-Y coordinate system starting with the upper-left corner of a page, then to the next page of the spread containing the XML, and so on. If you tagged XML content in exactly that order, this setting will work fine, as long as the required sequence of elements in the DTD for your XML and the order of the XML content in InDesign are the same.

If you check "Same As XML Structure", then your XML will be exported in the sequence it was tagged. Again, the content may or may not be valid, depending on whether that order matches your DTD, if you use a DTD.

In most cases, using the "Same As XML Structure" will provide a closer match to your DTD.

You can reorder XML structure to match the DTD in use in InDesign (or a different DTD) with an XSLT file applied during export. For example, if you need to get that <title> element always as the first element inside your root element, then the XSLT used on export can process the <title> element first, then go on to process other elements in the order that you need. XSLT can also rename, group, sort, add or rename attributes, wrap and unwrap XML elements and add a reference to a DTD or schema to your exported XML. Sometimes you will need to chain together several XSLT transformations to get the final XML that you want. See **Advanced Topics: Transforming XML with XSL**.

Advanced Topics: Transforming XML with XSL

Many of the topics in this section of the Short Cut deal with CS3's new features for dealing with XML, and the more painful ways to accomplish the same work using CS2's features.

An Aside Regarding Scripting InDesign and XML Rules-based Publishing

There is a lot of potential hidden in the phrase "XML Scripting", most of which applies to InDesign CS3. The power tools are Scripting (AppleScript, Visual Basic and JavaScript) in CS2 and CS3, and rules-based XML publishing (CS3 only).

Scripting InDesign promises to be another hacker's delight—for hackers who like publishing. If you want to get a better idea of what can be done, see the O'Reilly Short Cut publications, such as *Scripting InDesign with JavaScript* by Peter Kahrel.

You can get started with the scripts by visiting the Adobe web site and looking for "scripting reference" documentation in the InDesign Help and Forum sections of the site. Online searches will reveal more InDesign scripting references.

XSLT for Wrangling XML; XML Scripting for Automating XML Publishing

Here's what I think are the key differentiators for which process to use and why:

If you need to change the XML itself as it is being imported or exported into InDesign, use XSL transforms. I provide some examples in this Short Cut. Anyone can write XSLT* with a text editor and one of the good XSLT books; the trick is to know when to "speak InDesign" and how. InDesign has its limitations in transforming XML on import and export.

Note: While I provide working examples that you can copy, I will not detail every step that I take in writing the XSLT used in this Short Cut.

*eXtensible Stylesheet Language Transformation (XSLT) is written in XML and used to process one XML structure into another, or into a different format such as HTML.

If you need to use InDesign to handle tasks that are inherent to InDesign itself, such as **Map Tags to Styles**, you could use the XML scripting capabilities. Here are some of the scripts that are available:

javascript\xnnl

Go

x	Apply5stylesToXMLElements.jsx	^InnportXML.jsx	PlaceIntoCopy.jsx
A	^fj Apply Table5style, jsx	^fj ImportXMLIntoElennent .jsx	^iPlaceIntoFramne.jsx
	^fj AssociatedPageItem. jsx	^fj InsertTextAsContent, jsx	PlaceIntoInlineCopy, jsx
	^jj AssociatedXMLElement, jsx	i^LoadXMLTags.jsx	^PlaceIntoInlineFramne, jsx
	^jj Con vert AttributeToElement, jsx	^fj MakeProcessingInstruction, jsx	^PlaceXML.jsx
nlis	Con vertElementToAttribute. jsx	^MakßXMLAttribute.jsx	SaveXMLTags. jsx
i i	Con vertXMLElennentToTable, jsx	^fj MakeXMLComment .jsx	^iUntagElennent.jsx
	^jj CreateXMLElement .jsx	^MakeXMLTags.jsx	fISxXMLImportPreferences, jsx
	DeleteXMLElennent .jsx	^fj Map AllStylesToTags,jsx	^iXMLPreferences.jsx
	DuplicateXMLElement. jsx	^fj MapStylesToTags .jsx	iSJXMLStory.jsx
rfa(.)	ExportSelectedXMLElement, jsx	^ MapTagsToSty les. jsx	££xXMLViewPreferences, jsx
	^iExportXML.jsx	^iMarkup, jsx	
	ImportInto5electedXMLElement, jsx	^fj MoveXMLElement, jsx	

A list of InDesign XML scripts

If you want to try more advanced automation, there are scripts for what Adobe terms "rules-based XML publishing" as shown below:

javascript\xmlrules

Go

	ijAddAttribute.jsx	FindXMLElementByAttribute. jsx	ProcessChildren.jsx
A	^iAddReturns.jsx	^fj FindXMLElementByContent, jsx	^jj ReturningFalse, jsx
	^iAddReturnsAnd5taticText,jsx	^fj FindXMLElementByFindGrep. jsx	^fj 5ortXMLElements, jsx
	^attributes.xml	FindXMLElementByFindText, jsx	^TestingMoveElement.jsx
	^ Con vertTo Attribute, jsx	FindXMLElementByRegExp, jsx	^fj XMLRuleExampleSetup. jsx
	DepthFirstProcessing.xml	Forward AxisMatching,jsx	^fj XMLRules ApplyFormattling, jsx
	^fj DepthFirstProcessing_A, jsx	^fjglue code.jsx	⇒ XMLRulesChapterExamples. zip
	^ DepthFirstProcessing_B. jsx	^iListItems.xml	XMLRulesExampleData. xml
	^fj DuplicateXMLElement. jsx	^fj ListProcessing. jsx	XMLRulesLayout, jsx
	^fj ExtractSubset, jsx	^fj MoveXMLElement, jsx	^fj XMLRulesProcessor, jsx

A list of InDesign XML Rules scripts

The rules are intended for the high-end publishing workflows used by magazine, catalog, and newspaper publishers, but thanks to Adobe making them available, anyone can try playing with them.

It's beyond the scope of a Short Cut like this (which is already loaded with lots of good stuff) to explore everything we'll be able to do with Scripting and XML rules. Check the Adobe site for more technical documentation on these as new releases of InDesign roll out.

Now, on to what you might want to do with XML in InDesign using XSLT.

XSL: Extracting Elements from a Source XML File for a New Use

Sometimes, you know that there is content inside the main XML document that you would like to use for another purpose. The most common alternative purpose is something like making a list of figures or a table of contents from captions and titles in the XML file. But you can be more creative. Perhaps you need to find all of the keywords that were tagged in an XML file and extract them to build the foundations for a glossary XML file. Here is some XML content with keywords:

```
<Story name="Residency Requirements"
  <SectionHead>Residency Requirements</SectionHead>
  <para>New York State law requires that all students file proof of residence each academic year.
  For New York State residents, the proper form should be submitted upon registration. Until you comply
  with this requirement, you will be billed the <keyword>non-resident tuition rate</keyword> (twice the
  resident rate).</para>
  <topic title="Residents of Monroe County">
    <SubHeads>Residents of Monroe County</SubHeads>
    <para>If you have been a permanent legal resident of New York State for the past year, and a
    resident of Monroe County for the last six months, complete a Residency
    Certificate/<keyword>Affidavit</keyword>, sign it, and submit it with your registration.</para>
  </topic> [more...]
</Story>
```

glossaryTest.xml

An XSL transform can be pretty simple:

```
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:output method="xml" version="1.0" encoding="UTF-8" indent="no"/>
  <xsl:template match="/">
    <!-- make a new root element for the glossary XML -->
    <xsl:element name="glossary">
      <xsl:for-each select="//keyword">
        <!-- make an element for the term -->
        <xsl:elementname="term">Term: <xsl:value-ofselect-'.'7></xsl:element>
        <!-- make an element for the definition -->
        <xsl:element name="definition">Definition: </xsl:element>
      </xsl:for-each>
    </xsl:element>
  </xsl:template>
</xsl:stylesheet>
```

makeGlossary.xslt

The results will be very useful: a new glossary XML file, with keywords as terms, and a blank definition element following each term, ready for an author to work on. Because the keywords were in the source XML, and they are all now terms in the output XML, the transform makes it easy to correlate the glossary to the document that contains the keywords.

```
<?xml version="1.0" encoding="UTF-8"?>
<glossary>
  <term>Term: non-resident tuition rate</term>
  <definition>Definition: </definition>
  <term>Term: Affidavit</term>
  <definition>Definition: </definition>
</glossary>
```

glossaryOutput_XSL.xml

This type of XSL seems to work well on an existing InDesign file tagged with keywords. On export, if the XSL is used, the resulting file will just be the glossary XML, which you can put in a document of its own or into sidebars in a separate text flow in the same document as the original XML.

Alternately, you could export the entire document XML, and then apply the transform to the exported XML (using an XSLT processor such as Oxygen or XML Spy) to get your starting glossary XML file.

XSL: Getting the Elements to Sort Themselves

One of the common things that people who work with XML do is to sort XML content based on values of attributes on XML elements, or according to the text contained in the elements. Mostly, this is for tables (see the discussion on Tables elsewhere in this Short Cut), but it might be used for sorting blocks of content. For example, with this XML:

```
<?xml version="1.0" encoding="UTF-8"?>
<glossary>
  <term>Affidavit</term>
  <definition>A legal document that provides proof (of residency, payment, status as an emancipated
  minor, etc.). Affidavits must be prepared by an attorney, appropriate agency or counselor, and must be
  attested to under supplied with a notarization stamp.</definition>
  <term>F-1 Visa</term>
  <definition>An F-1visa is a nonimmigrant visa issued by the USCIS (formerly INS) to the alien student
  who is pursuing studies in the US. The "F" visa is reserved for nonimmigrants wishing to pursue academic
  studies and/or language training programs.</definition>
  <term>INS documentation</term>
```

«definition>Written materials provided by US Citizen and Immigration Services (formerly Immigration and Naturalization Services) to prove a student's status as an legal nonimmigrant.</definition>

<term>Non-resident tuition rate</term>

<definition>Students whose primary residence is not in the state will pay the tuition rate for non-residents. However, if a student resides within the state for the six months of the year immediately preceding the start of the session for which tuition is paid, the student qualifies for the resident tuition rate.</definition>

<term>EDU-PAY</term>

<definition>A payment plan for students who do not have the resources to pay the bill in full or who may not qualify for sufficient financial aid to cover the entire bill at the college.</definition>

</glossary>

glossaryInput.xml

You can use this XSL to write a top-level <glossary> element containing terms and definitions sorted in ascending alphabetical order.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:output method="xml" version="1.0" encoding="UTF-8"/>
  <xsl:template match="/">
    <xsl:element name="glossary">
      <xsl:for-each select="glossary/term"><xsl:sortorder="ascending"/> <xsl:copy-ofselect-'.'/>
      <xsl:copy-of select="following-sibling::definition[1]"/></xsl:for-each>
    </xsl:element>
  </xsl:template>
</xsl:stylesheet>
```

GlossarySort.xslt

This is the resulting XML; the definition for EDU-PAY is now near the top of the list of terms instead of at the end.

```
<glossary>
  <term>Affidavit</term>
  <definition>A legal document that provides proof (of residency, payment, status as an emancipated
  minor, etc.). Affidavits must be prepared by an attorney, appropriate agency or counselor, and must be
  attested to under supplied with a notarization stamp.</definition>
  <term>EDU-PAY</term>
  <definition>A payment plan for students who do not have the resources to pay the bill in full or who
  may not qualify for sufficient financial aid to cover the entire bill at the college.</definition>
  <term>F-1 Visa</term>
```

<definition>An F-1 visa is a nonimmigrant visa issued by the USCIS (formerly INS) to the alien student who is pursuing studies in the US. The "F" visa is reserved for nonimmigrants wishing to pursue academic studies and/or language training programs.</definition>

<term>INS documentation</term>

definition>Written materials provided by US Citizen and Immigration Services (formerly Immigration and Naturalization Services) to prove a student's status as an legal nonimmigrant.</definition>

<term>Non-resident tuition rate</term>

definition>Students whose primary residence is not in the state will pay the tuition rate for non-residents. However, if a student resides within the state for the six months of the year immediately preceding the start of the session for which tuition is paid, the student qualifies for the resident tuition rate.</definition>

</glossary>

You can apply sorts that create descending order simply by specifying `xsl:sort order="descending"` on the elements you are looping over with the `<xsl:for-each>`. For more about loops and sorts, see http://www.w3schools.com/xsl/el_sort.asp.

To sort before importing XML into InDesign, apply the sorting transform, and then import the resulting XML output.

Note: For large documents, sorting when importing the XML (applying the XSL transform within the **XML Import Options** dialog) may make InDesign crash. It would be better practice generally to do sorting as a preprocessing step.

Excluding some of the elements can be accomplished with placeholders and the XML Import Option **"Only import text elements that match existing structure"** (see the section on import options), but you might want to remove entire sets of elements, or only selected ones, on your way in or out of InDesign.

For example, let's suppose that you have product catalog XML that contains `<price>` elements that have both `<unit>` and `<bulk>` elements. *(I'm momentarily tired of course catalog XML. Periodically, I have some fun imagining weird products like environmentally friendly pet toys. It's late. Forgive me.)* You produce two catalogs: one for wholesalers and one for consumers. You might use the same XML file for each catalog, but suppress whichever price that doesn't apply in each catalog version.

Your XSL transform would look for the type of price you want in the catalog, and only copy those products for import.

This is the tiny example XML file, in which one element has a bulk price:

```

<?xml version="1.0" encoding="UTF-8"?>
<!-- <?xml-stylesheet type="text/xsl" href="bulkPrice.xsl"? ->
<products>
  <product type="petToy">
    <name>Chewy Toy</name>
    <catno>D123</catno>
    <price currency="USD">
      <bulk qty="100">.39</bulk>
      <unit qty="1">.99</unit>
    </price>
    <unit pack="recyled" measure="imperial">4 oz.</unit>
    <description>Delicious enviro-friendly dog treat made with processed pig parts, shaped like
miniature pig.</description>
  </product>
  <product type="petToy">
    <name>Squeaky Toy</name>
    <catno>D456</catno>
    <price currency="USD">
      <unit qty="1">1.49</unit>
    </price>
    <unit pack="recyled" measure="imperial">5.1 oz.</unit>
    <description>Exciting enviro-friendly dog toy made with recycled, nontoxic, heavy-duty plastics.
Squeak is air-produced, no hard parts inside. Safe for all mature dogs over 10 lbs. Not recommended for
puppies and smaller dogs.</description>
  </product>
</products>

```

petToys.xml

And here is a small XSL that looks for a bulk price and only copies <product> elements that have one:

```

<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:output method="xml" version="1.0" standalone="yes"/>
  <xsl:template match="/">
    <xsl:apply-templates select="products"/>
  </xsl:template>
  <xsl:template match="products">
    <xsl:copy-of select="product[price/bulk !='']"/>
  </xsl:template>
</xsl:stylesheet>

```

```

    </xsl:template>
</xsl:stylesheet>

```

The XSL produces this XML output:

```

<?xml version="1.0" encoding="UTF-16" standalone="yes"?>
<product type="petToy">
  <name>Chewy Toy</name>
  <catno>D123</catno>
  <price currency="USD">
    <bulk qty="100">.39</bulk>
    <unit qty="1">.99</unit>
  </price>
  <unit pack="recyled" measure="imperial">4 oz.</unit>
  <description>Delicious enviro-friendly dog treat made with processed pig parts, shaped like miniature
pig.</description>
</product>

```

petToyBulk_XSL.xml

In the example above, the Squeaky Toy item was excluded because it doesn't have a bulk price. When the XML is imported into the wholesale catalog with the XSLT applied during import, it will be pertinent to the wholesale audience.

Alternately, you could process the original XML with XSL as a separate step before importing the XML into InDesign.

Creating Wrappers for Repeating Chunks

Sometimes, you need to make a wrapper around part of the incoming XML to use the functions of InDesign placeholders or tables. With XSL, you can add an element to the original XML structure. Suppose you have a set of fairly complex elements that belong together, but they are just strung along one after another. In InDesign, you want to put each set into its own separate text flow. Here's an example of how you could make this happen with our previous glossary XML example:

```

<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:output method="xml" version="1.0" encoding="UTF-8" indent="no"/>
  <xsl:template match="/">
    <xsl:element name="glossary">
      <xsl:for-each select="glossary/term">

```

<!-- start the wrapper element inside the for-each loop so it will make a wrapper for each term and definition pair-->

```
<xsl:element name="glossTerm">
```

```
  <xsl:copy-of select="."/>
```

```
  <xsl:copy-of select="following-sibling::definition[1]"/>
```

```
</xsl:element>
```

```
<!-- end the wrapper element-->
```

```
</xsl:for-each>
```

```
</xsl:element>
```

```
</xsl:template>
```

```
</xsl:stylesheet>
```

Glossary_wWrapper.xslt

The resulting XML has a wrapper around each term and definition pair:

```
<?xml version="1.0" encoding="UTF-16"?>
```

```
<glossary>
```

```
  <glossTerm>
```

```
    <term>Affidavit</term>
```

```
    <definition>A legal document that provides proof (of residency, payment, status as an  
emancipated minor, etc.). Affidavits must be prepared by an attorney, appropriate agency or counselor,  
and must be attested to under supplied with a notarization stamp.</definition>
```

```
  </glossTerm>
```

```
  <glossTerm>
```

```
    <term>F-1 Visa</term>
```

```
    <definition>An F-1 visa is a nonimmigrant visa issued by the USCIS (formerly INS) to the alien  
student who is pursuing studies in the US. The "F" visa is reserved for nonimmigrants wishing to pursue  
academic studies and/or language training programs.</definition>
```

```
  </glossTerm>
```

```
  <glossTerm>
```

```
    <term>INS documentation</term>
```

```
    <definition>Written materials provided by US Citizen and Immigration Services (formerly  
Immigration and Naturalization Services) to prove a student's status as an legal  
nonimmigrant.</definition>
```

```
  </glossTerm>
```

```
  <glossTerm>
```

```
    <term>Non-resident tuition rate</term>
```

```
    <definition>Students whose primary residence is not in the state will pay the tuition rate for non-  
residents. However, if a student resides within the state for the six months of the year immediately
```

preceding the start of the session for which tuition is paid, the student qualifies for the resident tuition rate.</definition>

</glossTerm>

<glossTerm>

<term>EDU-PAY</term>

<definition>A payment plan for students who do not have the resources to pay the bill in full or who may not qualify for sufficient financial aid to cover the entire bill at the college.</definition>

</glossTerm>

</glossary>

gloss_wWrapper_XSL.xml

If you have a text frame for each glossTerm, you can drag each of the terms into its text frame.

Or, you can create a glossary as a text flow tagged as <glossary> that you set up previously for repeating block placeholder elements structured like this: an element called <glossTerm>, containing the <term> element, followed by a manual line break, then a tab and a <definition> element, followed by a paragraph break.

r=		r=	
& glossaryCS3.indt: glossary term glossa...		A	
[Basic Paragraph]	12	<glossTerm> Q term } gl 0 S5a ry	
	.	term(termn0	
	.	Q definition^ glossary	
	.	Clef in it i (<definition [J	
[Basic Paragraph]	.	(<glossTerm	

Placeholders (shown in Story Editor view) including the wrapper element that will be created with the XSLT

The <term> and <definition> elements are mapped to character styles to make them bold and italic, respectively. The <glossTerm> element is mapped to a Basic Paragraph style.

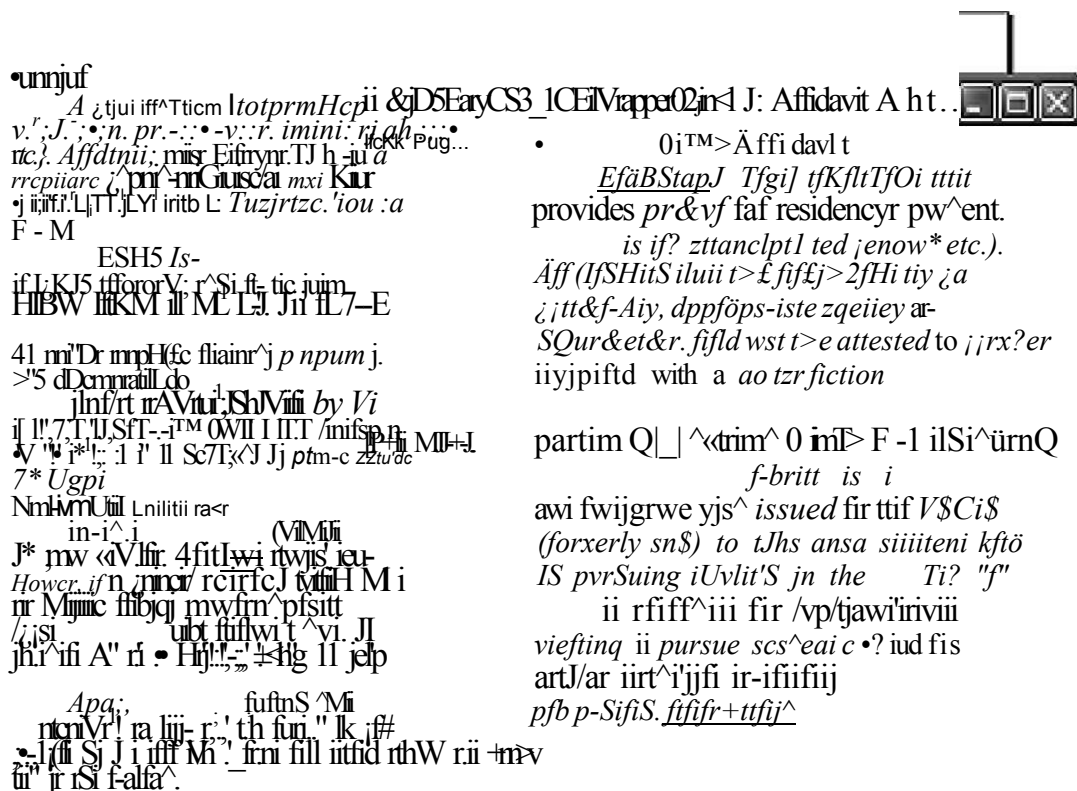
This is how the **XML Import Options** dialog should look, pointing to the XSLT:

XML Import Options

OptLaw	OK
Code: Macs Content v	f Cno?) 1
DenesteUie:	C3
<p>Pl^brenspe^t^fl tex; etenrenis</p> <p>DtSii/itport efemrYU that match e i i ^ structufe</p> <p>•licr,C" / GHF de-ffl&^s i^fl tÖCS if ÖÖS nÖCh</p> <p>[3]jj ret fimport contents of ivhfteipjcs-oif eienrievs</p> <p>••etetie eleiiiEnts, tranes* and cmfcant thai do rot unatch impated XML</p> <p>•ln^oil CAS libti^ii lnflc^i jai*B</p>	

Apply XSLT to create wrappers on import

The results will look like this in InDesign:



Example of creating wrapper elements with XSLT on import with placeholders for repeating blocks of content

By extrapolation, anytime you need to make a repeating block, the same XSLT technique can be used to develop richer XML with the required wrapper elements.

Making a Table from Element Structures

Now, let's use the same kind of concept to make an InDesign table from our incoming XML. In this example, suppose we do not want to have <Table> and <Cell> elements in our source XML because it violates our DTD. We can add the table markup solely within InDesign and keep our XML pure. We will use XSLT on import into InDesign.

We'll go back to the InDesign placeholder concept first and make a table that looks like what we want to generate (this is always the best practice if you need to make tables from XML in InDesign).

Our simple table looks like this: the table is tagged as <glossary> and it consists of a header row containing 2 cells tagged as <colHead> elements and another row containing one cell tagged as <term> and one tagged as <definition>.

f * il T) SKUE • •⁰... I... I: I_JL • I. J. I²... I: I_J • I^S... I, L. I.]²⁴... I • I
V [2) Ror

V ^₂ aoi-y

sj. («HuJ	Tina frL LL	ζsfiii^oi:
<u>Bp Nm</u> dtfinSion	r*H V QAdd Tig {7RVtg • ylijisTfiliM Qfrwt •Staff •TohlL Fjlcriii	[UllLj
	T .	v
	Si	3 .

Tagged table placeholder for the glossary

Note: InDesign doesn't use "rows" in its internal table model. So the table setup does not need anything tagged as <glossTerm> (which would map to a table row logically), as a container for the term and definition pair.

Now we use this XSLT on import by selecting it in the **Apply XSLT** option.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:template match="/">
    <xsl:element name="glossary">
      <xsl:for-each select="glossary/term">
```


If you set up the table with heading rows, they will repeat when the table flows from one text frame to another. If you use the alternate row coloring, you can achieve a look like this:

E	DeSiirk&	lenr	Défini noi}
SOMtfii	<i>A Icglt ifeetstraif ffoi jwwftffi proaf (cfresidency, QTiiHr.n,'.j. nu as an enxsneipasedmnor, ••j'). Affidavits hviJ' be prepared 5v S amirjitfv. appropriée agenzv or rmuKsloK and nws! te anmea h? wppHid v, r-fi a wtor&Mim JROW).</i>	N&ro-kffidfic d- 11 • [U] U [t	is not fâ .-ne stare n ill pay jhe m- iTioa mtsfar HWiniAui. H'j'H- ever; if c smtfeni resides nfir.w its siatefor the sir »tanHa cf; ln year immadiatefypreceding the star i eftie. ieiürt> f\$y i + àir.Ji mî-ifr H paid, ri.t}Mdf>i! ûr.iJyi« .101 jV WJidjff ruirfin jLin.
F-L Visa	<i>An F-iWifi " .t iwfflfoimcgranî visa wrawf in- ire USCIS {forrnBriy m to the aiiin jarcfmfwfa? rj JMRwng-iftsiai in iifi UK TTP "F " lijiii is reier. edfoylfDMivjBrf- ffV/tf V, tshtlf 79phKie JKfdmC ζwrites anà/orimjaoga imVj¹ programs.</i>	EDUFAY	<i>A payaientploit forifudwih AIC do not ptiJvs the resfures to pm the kti! t'tt fill CT gmn PF qualify fvr suffitrtn:financial aid cover-jhe enriva bili a\ jhe cel- JfW- •lijlbibJf]</i>
INS ([«Jui fn- tntdcu	<i>iiifJITJ m flflf T ^ I 01 TM OT Offiiifw andlmnjp'atiuri SijvIVj; iYUIMV IWJICMa>J Zpf YfU- riTiriLli-roii .Sendee^? to prove a jii'ar i iitiilij ti gf iega': nwin- PigikTii.</i>	(enn t W	

Imported table flowed into multiple text frames

See the **Help** section on Tables for more information on creating the look you want in your table layout.

Note: If you want to use tables to make run-in heads (similar to how many people do it in MS Word™), just turn off the rules and shading in the table styles in InDesign.

To make true run-in heads in InDesign CS, you should follow the directions in the **Help** for **Nested Styles**.

Upcasting Versus Downcasting

In XSLT, changing simple XML or HTML to a more complex XML form is "upcasting," while changing XML to a simpler DTD is "downcasting "

It may be that you need to convert HTML content into some form of XML. The simplest way to do this is to save your HTML as XHTML, a form of HTML that conforms to XML rules. Once it has been cleaned up as XHTML, you need to change the file extension to **.xml** before you import it into InDesign.

Note: You can use an open-source utility application called HTML Tidy (http://tidy.sourceforge.net) to make valid XHTML from your web page content. It is also bundled with Adobe Dreamweaver and some other applications that you may be using. Check for the ability to save web content as XHTML or HTML 4.0, also known as "strict" HTML.

It is possible to import the XHTML file without any transformation, but we want to modify the incoming XML somewhat to make it more compatible with our InDesign layout.

We will model the XHTML elements that correspond to the InDesign tagging concepts. We don't necessarily need the <head> element, because it is used only for the HTML title bar and metadata, not for anything that we will print. So we can make a structure like this in InDesign:

<p>fi idilnilCSa.indd: teil xMml blUly corit,.. - !f X</p> <p>itt rtliiml bodv COJH4jll (ltd be-nJ)</p> <p>i</p> <p>i table ccli (tdji fable «li (tdj</p>	<table border="1"> <tr> <td data-bbox="667 1117 837 1774"> <p>(itrt rv+i... fb>* Hf- w Citt P+g- Dp*gn (SirA PiiJ-j... Pv+i.-. BH* H+Qm C&f •• f*+W PjrtQ,-</p> </td><td data-bbox="837 1117 1316 1774"> <p>Jott tDf ipnlint ini hflidJ</p> <p>ET^Q^i^1«*!]: Hi) P1®</p> <p><£13</p> <p>tafele cöli</p> <p>iflll</p> <p>tt.3) j ü H ^ ^ n</p> <p>V</p> </td></tr> </table>	<p>(itrt rv+i... fb>* Hf- w Citt P+g- Dp*gn (SirA PiiJ-j... Pv+i.-. BH* H+Qm C&f •• f*+W PjrtQ,-</p>	<p>Jott tDf ipnlint ini hflidJ</p> <p>ET^Q^i^1«*!]: Hi) P1®</p> <p><£13</p> <p>tafele cöli</p> <p>iflll</p> <p>tt.3) j ü H ^ ^ n</p> <p>V</p>
<p>(itrt rv+i... fb>* Hf- w Citt P+g- Dp*gn (SirA PiiJ-j... Pv+i.-. BH* H+Qm C&f •• f*+W PjrtQ,-</p>	<p>Jott tDf ipnlint ini hflidJ</p> <p>ET^Q^i^1«*!]: Hi) P1®</p> <p><£13</p> <p>tafele cöli</p> <p>iflll</p> <p>tt.3) j ü H ^ ^ n</p> <p>V</p>		

Placeholder for XHTML import

Remember that you can use XSLT to change the order of elements (such as sorting alphabetically), but this is best done as a preprocessing step before importing XML, rather than using the **Apply XSLT** option.

Note: To make the import process format imported XML text automatically, create paragraph styles that match the names of the XHTML elements (h1, p, li, etc.) and use **Map Tags to Styles** with the **Map By Name** box checked to apply the paragraph styles to the placeholder elements before you import the XML.

To make the XHTML <table> element, which is always a lowercase word, match to InDesign's internal tag <Table>), which is always uppercase, we will use the Tagging Presets dialog on the Tags panel. We also want the <td> element to map to the InDesign <Cell>.

Tagging Preset Options

Options

Text Frames:	Story
Tables:	table -
Table Cells:	td -
linages:	Image -

OK

Cancel

The Tagging Preset Options of the Tags panel, setting Tables and Table Cells to <table> and <td> elements

In my tests, I used a simple XHTML file that looked like this:

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/xsl" href="xmlizeXHTML.xslt"?> ->
<html>
  <head>
    <title>XHTML example</title>
  </head>
  <body>
    <h1>An example of XHTML</h1>
    <p>Some general rules for XHTML are</p>
    <ul>
      <li>Every start tag must have a matching end tag</li>
      <li>All tag pairs must end without crossing over other end tags (to create properly nested
structures)</li>
```

Tag names cannot start with a number, and they cannot include any spaces, or "illegal" characters, such as ? and /, which can be confused with parts of the markup and processing instructions.

<table border="1">

<tbody>

<tr><th>a table header (th)</th></tr>

<tr><td>a table cell (td)</td></tr>

</tbody>

</table>

</body>

</html>

XHTMLsample.xml

The XSLT that we will use to simplify the XHTML looks like this (but this example is not developed to handle all possible XHTML elements):

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:output method="xml" version="1.0" encoding="UTF-8" indent="yes"/>
  <xsl:template match="/">
    <xsl:apply-templates select="html/body"></xsl:apply-templates>
  </xsl:template>
  <xsl:template match="html/body">
    <xsl:element name="body"><xsl:apply-templates/></xsl:element>
  </xsl:template>
  <!-- copy some elements directly -->
  <xsl:template match="h1|h2|h3|h4|h5|h6|p|ul|ol"><xsl:copy-of select="."/></xsl:template>
  <!-- simplify the table structure to what InDesign uses, no tbody or tr elements needed -->
  <xsl:template match="table">
    <xsl:element name="Table"><xsl:apply-templates/></xsl:element>
  </xsl:template>
  <xsl:template match="tbody"><xsl:apply-templates select="tr"/></xsl:template>
  <xsl:template match="tr"><xsl:apply-templates select="th|td"/></xsl:template>
  <xsl:template match="th"><xsl:element name="Cell"><xsl:apply-templates/></xsl:element>
</xsl:template>
  <xsl:template match="td"><xsl:copy-of select="."/></xsl:template>
  <xsl:template match="img">
```

```

    <xsl:element name="Image"><xsl:attribute name="href"><xsl:value-of
select="@href"/></xsl:attribute></xsl:element>
  </xsl:template>
  <!-- exclude the head tag content -->
  <xsl:template match="html/head"/>
</xsl:stylesheet>

```

xmlizeXHTML.xsl

We can import the XML and apply the XSLT to it as it comes in. This will strip off the unnecessary <head> element and simplify the <table> by removing the <tbody> and <tr> tags. Select the <body> placeholder element in the structure view and use **File>Import XML**, then select the XHTML file that you saved with a .xml extension.

Note: The import operation crashed sometimes when I selected an <html> element in the placeholder as the element to import into. Importing worked OK when I selected the <body> tag as the location to import the XML. This may be because we are creating an XML file that uses the <body> element as its root. At any rate, be forewarned that importing and applying XSLT can be fraught with peril—save a copy of the file with placeholders or make it an InDesign template before you start applying XSLT while importing XML.

The settings for the **XML Import Options** dialog will be: **Apply XSLT**, **Clone repeating text elements** for the ul/li structure, **Only import elements that match existing structure** (*it is important to check this*), and **Do not import contents of whitespace-only elements**.

XML Import Options

Options

Mode: Clone

Use Create It*

OK]

[Cancel]

HTML as XML Import Options

Import HTML as XML

Import HTML elements that match existing Structure

Effect: test elements to tables if tags match

Import contents of whitespace-only elements

• Delete elements, frames, and content that do not match imported HTML

Import HTML as XML in InDesign tables

Settings for the XML Import Options dialog for the XHTML-as-xml import

The results, after some tinkering with the Paragraph styles:

■ n example qi HhLCV-Mnt Ee Khhl body mill...

HTML

firm 11 kr r• HL

Enr UHl M hm i MitMt 4M Lj+ t

A3 m n im M MivehU M U^ M' .

Sl* H HBW m M itw

(iHi)

r • I - L H M laj (n li

• UAH riM l«;- b utii Ml iia.

irih^iBii ibaii

rh * ijhi'i i by lmsp

^ Lu^ • HiirJir

m* KiiiHiLuiLuuuiHunnm - x

l f

^ J U I

u

u

n

ki

14

u

• J jj £

•XBIfU <ifttTHLflfg

EJr>SC>e flf^rl r||l«. fgj HHI Sr<

atirt t^i wit hjvt a

•itthiri i-ni 1jg auch »

•^^ip") ^iQiaga around a plani

pir JV

>>>11 tij pi» rs iutt snd <thDUt

r.i ov«r <Ch> End Ijji [En cr-ralc

propf-ly init>d 5trgfty>ii,

Jl iyiiif^ii cf Ipurnid fmsHm la

•!/? j3t^1 til ^-niirj j-Pfr

pafagf^h tuL <ft ha> >n

l raiir. imiioi iLjrL with a nuitdr .

jr.] tm <annot lidli& any ipacfs. ur

Jmi|j| " tbaratli r4, 4(Till

> Th^h g anfl E---!)

em bd confuted with pjrts di th> <jmif

jmi prKtsmng InstructiQnj. (iQ

-iil HP>J^ THNL iantdi IB

J |J ft* <tpdr>fro*|1 flliitfit, hiich tr

tym caritalni tri [j i rllii>

aliBfnH infl 44kdtl<fii I

fht5dir>3) fii>intiL The

____jgbQdy>ff^ n fciy be fblCwtd by i

fT^' ilibl is vijpp 6 r.

ri.vjm uilH i I

XHTML imported into InDesign and formatted with matching paragraph and character styles

Upcasting from HTML to XML for InDesign Import

You can extend this concept to all the tags in the official XHTML DTD if you wish. Generally, you would want to use XSLT:

- To remove unnecessary structure that InDesign doesn't use (like the <head>, <tbody> and <tr> elements)
- To wrap elements that you want to have treated as repeating blocks (like or elements that contain elements)
- To change names of elements to match InDesign's built-in names (such as Table and Cell)

Downcasting to HTML

In some cases (who would ever want to do this?), you might want to save some XML in InDesign as web page content that matches a particular CSS. You can do this by saving the content with an XSLT written to downcast the XML structure to standard HTML tags. (See also the section **Exporting XHTML when XML is in your InDesign file.**) Let's try the XSLT export to HTML with the course descriptions that we have worked with in the early parts of this Short Cut.

First, we need to decide what the output for the HTML will be. We'll pretend that we have a CSS already set up for displaying the course descriptions online. To make it simple to understand, the CSS classes have names that correlate to the XML elements.

Note: I recommend you always use class names that will help you remember the XML element origin of the HTML output. If you use solely the <h1>, <h2> and <p> tags, later you will not be able to differentiate the HTML produced from <CourseDescriptions> from any other HTML except by reading the text for meaning. These types of semantic class names also may have benefit to help search engines as HTML "microformats" become more widely used.

We are going to create a new XHTML output from the course content using a series of steps. We can use the same tags that we created for the XHTML example for this example. We already have a simple tagged file with <h1>, <p>, , , <p> and <code> elements in a <body> element, which we can export to get an idea of the result file that we want. Select the XML in the **Structure** pane, then use right-click to get the **Structure** menu and select **Export XML**. In the export dialog, use these settings:

Export XML

General images

Options

☐ Include DTD Declaration

☒ Use XML Using iWillcoexs

☐ Export From Selected Element

☒ Export Untagged Tables as CALS XML

☐ Remap Break, Whitespace, and Special Characters

☒ Apply XSLT: Use Sc if. in ML J

Encoding: UTF-8

The Export XML dialog with options set to make quasi-XHTML output

Note: The box for **Remap Break, Whitespace and Special Characters** must be checked to clean up typographic characters that will not display in a browser properly (line returns, pretty quotes, em dashes, ampersands, etc.)

After you export, change the file extension of the exported file from .xml to .html and see how it looks in your browser. (For our example, which did not have a <head> element, you would need to edit the output file if you want to make it conform to the HTML strict DTD. Check w3cschools.org for good tutorials on XHTML.)

Ail example of XHTML

Some general rules for XHTML are

- Every start tag must have a matching end tag such as <p>tags around a plain paragraph.
- All tag pairs must not cross over other tag pairs (to create properly nested elements). An example of a simple XML document is given below, ending after a paragraph.
- Tag names cannot start with a number, nor they can contain any spaces, or "illegal" characters, such as ? and /, which can be confused with parts of the markup when processing it.

Example: A simple XML document

An XHTML document contains a root element which in turn contains a <table> element and a <caption> element. The <caption> element may be followed by a <caption> element inside the <table> wrapper.

The XHTML export (cleaned to conform to the HTML strict DTD)

Compare the course content to the HTML tags to see how you might want to tag the final output. For example, you might want to output the element <CourseDescriptions_Major> as an <h1 class="major">, and the <CourseDescription_Name> as an <h2 class="coursename">, the <CourseDescription_Text> as <p class="coursedescription">, and the prerequisite <CourseDescription_Note> as <p class="coursenote">. The semantically meaningful CSS class names will help you be clear about the subject of the content, and where it originated.

This is the sample XML:

```
<CourseDescriptions>
  <CourseDescription_Major>Accounting</CourseDescription_Major>
  <CourseDescription>
    <CourseDescription_Name>ACC 101 Accounting Principles I 4
Credits</CourseDescription_Name>
    <CourseDescription_Text>Basic principles of financial accounting for the business enterprise with
emphasis on the valuation of business assets, measurement of net income, and double-entry techniques
```

for recording transactions. Introduction to the cycle of accounting work, preparation of financial statements, and adjusting and closing procedures. Four class hours.</CourseDescription_Text>

<CourseDescription_Footnote type="prereq">Prerequisite: MTH 098 or MTH 130 or equivalent.</CourseDescription_Footnote>

</CourseDescription>

...more

<CourseDescription_Major>Art</CourseDescription_Major>

<CourseDescription>

<CourseDescription_Name>ART 101 Art Essentials 3 Credits</CourseDescription_Name>

<CourseDescription_Text>This course is designed to improve the student's visual perception and expand critical awareness through a variety of hands-on studio projects. The student will become familiar with the methods, materials, media, vocabulary, and techniques of making art. This course is suggested for students who are interested in developing their creative skills but are not art majors. Two class hours, two studio hours. (SUNY-A)</CourseDescription_Text>

</CourseDescription>

</CourseDescriptions>

CoursedescriptionsOl.xml

The XSL transform could accomplish these XML to HTML changes: generate a valid <html> with <head>, <title> <link> (for the CSS file) and <body>, and then make the individual heading levels with the CSS classes and even some ids. This is a simple XSLT without any loops:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:output method="html" encoding="utf-8"/>
  <xsl:template match="/">
    <html>
      <head>
        <title>Course descriptions web version</title>
        <link rel="stylesheet" href="courses.css" type="text/css"></link>
      </head>
      <body>
        <xsl:apply-templates/>
      </body>
    </html>
  </xsl:template>
  <xsl:template match="CourseDescriptions">
    <h1>Course Descriptions</h1>
    <xsl:apply-templates/>
  </xsl:template>
```

```

</xsl:template>
<xsl:template match="CourseDescription">
  <div>
    <xsl:attribute name="id"><xsl:value-of
select="substring(CourseDescription_Name,1,7)"/><xsl:apply-templates/></xsl:attribute>
    <xsl:apply-templates/>
  </div>
</xsl:template>
<xsl:template match="CourseDescription_Major">
  <hr/>
  <div class="major">
    <xsl:attribute name="id"><xsl:value-of select=".7"></xsl:attribute>
    <h2 class="major">
      <xsl:value-of select="."/>
    </h2>
  </div>
</xsl:template>
<xsl:template match="CourseDescription_Name">
  <h3 class="coursename">
    <xsl:value-of select="."/>
  </h3>
</xsl:template>
<xsl:template match="CourseDescription_Text">
  <p class="coursedescription">
    <xsl:value-of select="."/>
  </p>
</xsl:template>
<xsl:template match="CourseDescription_Footnote">
  <p class="coursenote">
    <xsl:text>Notes: </xsl:text><br/>
    <xsl:value-of select="."/>
  </p>
</xsl:template>
</xsl:stylesheet>

```

courseDescriptions2HTML.xsl

The CSS has only a few classes in it:

```
body
{background-color: #ffffff;}
h1, h2, h3 {font-family: Verdana, Arial, sans-
serif; ;margin-bottom:6px}
.major {color:green; font-size: 19px;margin-bottom:0px}
.coursename {color:navy; font-size: 13px;margin-
bottom:0px}
p {font-family: Arial, Verdana, sans-serif; font-
style:normal; font-size: 13px}
.coursedescription {font-
style:normal; color:#000000;margin:0px 0px 0px 0px}
.coursenote {font-style:italic;color:#333333/margin: 0px 10px
20px 20px}
```

Courses.css

The HTML output looks like this (viewed in the browser to see the CSS applied).

```
m* 4tnfi#rtan ntb Trnb* lflhrmcjl lnWHl l ifk0lll
[A          Wt »tfj
      •      jil]      pswi •      G3 • ^ S JÛ « i
                                           ¿3«
```

Course Descriptions

Acceuhthfg

```
A« III L ACHIMM HJilich I 4 LUclh
9mC filpilh] allatmid KC Ong fiw l* thtrnd *N+H4 ml ih'uid ù h> tuHpiñ G hrtttfi Idltfl JtH imitij <A hi:
ihLil Un fraeMhij Hihhihij I I -I KOnal lmtHijlir lllHQQn 14 lh* L filr (H ELJfep] n4ll. ptpM Fil mtaVKil
-ùHrvxt. JA i^jtüc tML totitgFui^cltnr-nih
      M M
      F i B ^ fl Û fl wttil tO & tefMittf
À<V IH ACCHIMM) :lH>Jld 11 4 Cwfttl
r r i -jil- l r f l* blut pntplH cThi' n Ktlur») -uluInj J Lud opilM Hiyi tñj EMFñH HElnil TU ¿uMclh
vr* Jh dñ Slpñllil rECCUñt Ukdy wh Lili+ñt FkMplñ tñkVH lll ' lñ^b jilFj] dJ utt in UN • n vd
¿Lldñtj bulOE Unfm Hr< lME Kua
ifclIP*
F Î I H * OfC Sr • H 10 i'U -407 1 ?1 -Mñ hi nt/tf? iV <~.V' <~ñt
```

An tñk • 4j Nhj Ihj II V

```
.HI ^nCimfiirTV l IfñM 1
An iññ+jññi jr. lñ> in jññppñr-jñ wh mclñnc i i n:hiniñilr n-rphñlpl qññi-pñr-jv òpñpp ungi <ñ hTiñ
Lr
```

XHTML created from XML exported using Apply XSLT to add CSS styles

Notice that the course numbers have been used to generate unique IDs for the <div>s containing course descriptions:

```

<html>
<head>
<META http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>Course descriptions web version</title>
<link rel="stylesheet" href="courses.css" type="text/css" />
</head>
<body>
<h1>Course Descriptions</h1>

<div class="major" id="Accounting">
<h2 class="major">Accounting</h2>
</div>
<div id="ACC 101">
<h3 class="coursename">ACC 101 Accounting Principles I 4 Credits</h3>
<p class="coursedescription">Basic principles of financial accounting for the business enterprise with
emphasis on the valuation of business assets, measurement of net income, and double-entry techniques
for recording transactions. Introduction to the cycle of accounting work, preparation of financial
statements, and adjusting and closing procedures. Four class hours.</p>
<p class="coursenote">Notes: <br>Prerequisite: MTH 098 or MTH 130 orequivalent.</p>
</div>
...more

<div class="major" id="Art">
<h2 class="major">Art</h2>
</div>
<div id="ART 101">
<h3 class="coursename">ART 101 Art Essentials 3 Credits</h3>
<p class="coursedescription">This course is designed to improve the student's visual perception and
expand critical awareness through a variety of hands-on studio projects. The student will become familiar
with the methods, materials, media, vocabulary, and techniques of making art. This course is suggested
for students who are interested in developing their creative skills but are not art majors. Two class hours,
two studio hours. (SUNY-A)</p>
</div>
</body>
</html>

```

coursedesc.html

Note: To get HTML, I used the XSLT on export, but it took a few trials to get it to work. It generated some error messages that require some fairly deep understanding of XSLT to fix. It sometimes crashed if I tried to do anything too fancy with sorting or rearranging the XML. And, I could only name the exported file with the ".xml" file extension.

However, Adobe has some nice features part of the XHTML export. Check the online documentation, including videos, that Adobe provides about XHTML export from InDesign. One of the options is to add empty CSS classes to content as you export it. Then, you could add the kind of semantic names that I am suggesting if it is worthwhile to help you remember the source of the content. Or, you can use a CSS that you already have for your web site and forget the semantics. So if XSLT is too daunting, why bother? Just create XHTML directly. Thanks, Adobe. (I think. You won't get to do all the cool things that XSLT can offer, but that may be overkill for a lot of folks. Just because I love XML and XSLT doesn't mean that everyone else has to suffer for it.)

Generate a Link with XSLT (Not Automated)

Have you noticed that I haven't mentioned anything about links in my discussion of generating XHTML with XSLT? Adobe has enabled an auto-generation of hyperlinks in InDesign (if you use the **Cross Media Export>Export to XHTML**, I believe the <a> tag will be created from a hyperlink you have created with a URL). But, if you are writing your own XSLT, there is no built-in method to get the <a> tag in the **Export XML** output. The most straightforward way to handle it is to create a standard web <a> tag in your content, give it an href attribute, and export it.

If your current InDesign XML tag set doesn't include the <a> tag, you can add it to the list of tags. In the **Tags** window, select **New tag**, then overtype the highlighted tag name in the window with "a." Switch to the **Text Selection** tool, select the text you want to use for your link, and tag it by clicking the new <a> tag name in the Tags list.

You must add the required **href** attribute to the <a>-tagged text. With the tagged text selected in the **Structure** pane, click the round black button in the top of the **Structure** window. When the **Add Attribute** window opens, type "href" (without quotes) in the **name** text box and the destination URL in the **value** text box.

To get this element in your export when you are using Apply XSLT, just include a small template that copies the <a> tag, including its attribute.

```
<xsl:template match="a"><xsl:copy-of select="."/></xsl:template>
```

A simple XSL template to copy <a> link tags

With XSLT, you can also rewrite an existing element such as `<link>` to an `<a>` element with an empty `@href` attribute. Then you can edit the `@href` in an HTML editor to insert the correct link path and file name. More advanced XSLT can use automation to pass in a relative path to a known web site location and pre-pend it into the `@href` value. For example, you can have a parameter in your XSLT, `"$mysite"`, defined as **`http://www.mysite.com`**, then insert that as the starting part of every `@href`. Then all you would have to do is add the correct filename for the .html page or other resources that your `@href` points to.

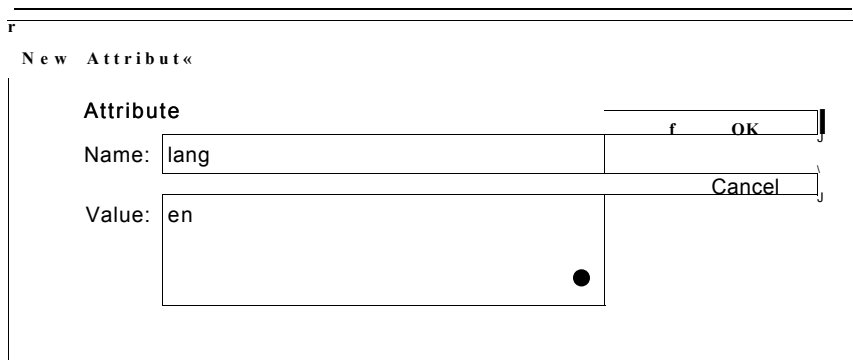
Adding Useful Attributes to XML

Attributes are little descriptive bits of information used by programs and people to find and manipulate items in XML files. You can add attributes to indicate

- the status of development (draft, revision, final, approved)
- the audience, or type of user who would want a piece of content (novice, expert, user, service operator, etc.)
- the currency of monetary values
- or whatever will help you work with your XML content.

Generally, it is easier to add attributes with XSLT than within InDesign, if you want to use them all over your XML, but Adobe gives you a tool to add an attribute manually, so we'll start with that.

Just select an XML element in the **Structure** pane, then click the big black dot in the top icon bar of the **Structure** pane (or right-click and select **New Attribute** from the Structure menu). In the **New Attribute** dialog, enter the name of the attribute and the value.



The New Attribute dialog used to add the lang attribute

Note: IMHO, Adobe should have included the ability to add standard attributes for `lang`, whitespace handling, and other attributes that people who work with XML commonly use. Because these are not available with any kind of menu or picker in InDesign, you have to add them manually, or with XSLT.

A general formula for adding attributes

Here is a general formula for XSL templates to add an attribute to your XML:

```
<xsl:attribute name="attname">attvalue</xsl:attribute>
```

where **attname** is the name of the attribute and **attvalue** is the value you want to assign to that attribute. In my course descriptions XSL example, I used a generated value in several places. This template takes a part of the text string (characters 1 - 7) that is contained in a `<CourseDescription_Name>` and uses that as the value of the id attribute that is being added to a div, such as `<div id="ACC 101">`.

Note: The substring operation seemed crash-prone when I used it with **Apply XSLT** when importing XML. It might be better to apply the XSLT to the XML as a preprocessing step when you want to do string operations with your transforms.

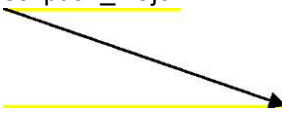
```
<xsl:template match="CourseDescription">
  <div>
    <xsl:attribute name="id"><xsl:value-of
select="substring(CourseDescription_Name,1,7)"></xsl:attribute>
    <xsl:apply-templates/>
  </div>
</xsl:template>
```

An XSL template to capture a substring (selected part of element text)

Generating an id attribute for a div

In the next template, the value of the `<CourseDescription_Major>` element is being used to generate the div's id value, such as `<div id="Accounting">`. The "." (dot) in the `xsl:value-of select="."` will be replaced with the word Accounting (the dot notation means "self", or the current element's content).

```
<xsl:template match="CourseDescription_Major">
  <hr/>
  <div class="major">
    <xsl:attribute name="id"><xsl:value-of select="."/></xsl:attribute>
    <h2 class="major">
      <xsl:value-of select="."/>
    </h2>
  </div>
</xsl:template>
```



An XSL template to generate a div id from element text

Use of the lang attribute for translations

For the sake of multilingual content, you might want to add an attribute of "lang" to the root element of your XML file. The general syntax for the attribute is `<elementname lang="en">` (or `lang="es"` for Spanish, `lang="de"` for German, `lang="it"` for Italian, etc.). It can be important to know whether the content is American-English or British-English (for instance, to know what the measurement units should be), so the approved country codes can be appended to the lang code, as "en-us" or en-gb."

If you are writing content that mixes in some words, phrases, or quotations in other languages, you might think about marking these up as inline elements (keyword, glossterm, or some other element), and using a lang attribute to indicate the content is in a different language. For example, Latin names in scientific or medical texts, Spanish song titles in a music catalog, or French words in a cookbook, all could be treated in this manner.

Tip: If an entire block element like a `<p>`, `<div>`, or `<table>` should be marked as a different language, put the lang attribute on the element, rather than adding an inline element with a lang attribute.

Conversely, if there is text that you do not want to have translated, such as company names, slogans, trade, and brand names, you could use `lang="en"` and add another attribute such as `translate="no"` on the same word or phrase, such as `<keyword lang="en" translate="no">Textenergy LLC</keyword>`.

Note: If you think ahead about these issues at the time you create your XML, you can save a lot of money on translations costs.

Creating an image href attribute

If you want to import graphics, the XML element that will contain your image needs an attribute named "href" and the path and file name of the graphics file.

The XSL transform is pretty simple: whatever the name of your element (`<Image>`, `<myBanner>`, `<logo>`, or whatever), you just need put code in the template for that element to append the href attribute to the element.

```
<xsl:attribute name="href">file://path/filename.ext
</xsl:attribute>
```

Here, **href** is the *required* name of the attribute, and you replace the `file://path/filename.ext` with the path (can be many folders long) and the filename of your graphics file. You can even leave the basic XSL template with this boilerplate `file://path/filename.ext` in it; it will create a valid placeholder for the image and then you can fill in the correct path and filename

later if you don't know it at the time you create the XML for import. (This is often the case when someone says that you will have artwork for certain places in your document, but he hasn't provided the actual images yet. Once you know which image goes where, and what the path and filename is, adding images into your XML is as simple as writing the correct href value.)

Pathing to images

If the folder where your images are located is known, you can write the path as part of the XSLT template. Just replace the path boilerplate with the real path, so it reads as

```
<xsl:attribute
name="href">file://C:/test/images/filename.ext
</xsl:attribute>
```

or whatever drive and folders you need to point to. Just realize that you are hard-coding it in, so if you move the graphics to a different location, the links will break, and the image won't show up in the InDesign layout.

If you plan to export your InDesign file later, you should use the image files' relative pathing such as

```
<xsl:attribute name="href">file://images/filename.ext
</xsl:attribute>.
```

(In the example above, the `images` folder is below your InDesign document folder, but it should be whatever your actual relative path is to replace `//images`.)

A Word about Using Find/Change for XML Markup in InDesign

A feature introduced in InDesign CS3 is the ability to search and replace on the characteristics of text. You can use **Edit>Find/Change** as a means of locating specific types of styling in your document that you would like to mark up as XML elements. The process is described in the Adobe InDesign Technical Reference documentation, available on the Adobe web site. The **Find/Change** dialog has an expression language that lets you write a pattern to search for and a pattern to replace the found text. Using the **Find/Change** dialog, you can select an XML element to apply from the Tags list (the tags must already be available in your InDesign document, either from using **Load Tags** or **Load DTD**).

This method may be very helpful for applying or fixing up XML markup, as when you decide to change the name of an XML element after it has been applied. (I would not recommend using **Find/Change** if you are working with a *linked* XML file that you have imported into InDesign.) At the least, save a copy of your InDesign file before using **Find/Change** to alter or add XML markup.

A Brief Note about InCopy and XML

According to James Maidvald,

"InCopy can:

- Open an InDesign document to edit or add XML structure.
- Open an InDesign assignment or InCopy .INCX file to edit or add XML structure.
- Open an XML file directly.
- Create an XML file from scratch."

Maidvald's book is *Designer's Guide to Adobe InDesign and XML: Harness the Power ofXML to Automate your Print and Web Workflows* and can be viewed here: <http://my.safaribooksonline.com/9780321549914>

Note: Since version CS4, the .icnx file format has been replaced by the .icml file format, for interchanging InCopy files with CS4 and CS5. See **About IDML and ICML**.

InCopy files externalize InDesign text flows for copy editing without using InDesign. The licenses for InCopy are much cheaper than the ones for InDesign.

A designer assigns an InDesign text flow to an editor or writer as an assignment file. The writer or editor checks out and opens the assignment in InCopy, where the InDesign layout for the text flow can be seen, but not changed, by the person working on the text. Typically, the writer or editor can apply color swatches, make text bold, italic, bulleted, etc., and apply approved paragraph and character styles.

Where it gets interesting is when you want to create XML files, but you don't want to have all the writers trying to make XML content in the same InDesign file. In this case, you can create XML files using a DTD in InCopy just as you would in InDesign. So each writer can create XML in their assigned text flow using the Story editor with tags viewable, and switch to the other views to see how the text looks and whether it is overset for the assigned text flow. (There's also a view in InCopy that will show exactly the word count and line breaks of the text flow).

You can save InCopy file as .icml (the default format), or if you have tagged them, you can save them as .xml files. The following example is XML tagged content in InCopy, styled with paragraph styles and exported as RTF, followed by its tagged content in XML.

Note: Neither InDesign nor InCopy allow you to tag footnote content as XML elements. This seems to be a bug, not a feature, since when you export XML, it seems probable that you would want your footnote content exported, too. But your footnotes are nowhere to be found in the exported XML.

a < /ti/t^/r. - /fa'y

by inline byline, author_name element with style authorname

A. paragraph of p element tagged text in the body of the story, which uses no DTD-defined XML, just *ad hoc* tags. All line breaks occur outside of the XML tag. This paragraph, of style p_first, is honored by its stylish drop cap.

Another paragraph tags along with a footnote.⁴

Footnotes can't be tagged in InDesign, and that's the end of the story.

An InCopy file saved as RTF for import into Word

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<Story><title>A Tale of a Simple, Tagged Story</title> <byline>by
<author name>inline byline, author name element with style
authorname</author name></byline><content> <p>A paragraph of p element
tagged text in the body of the story, which uses no DTD-defined XML,
just <emphasis type="foreign term">ad hoc</emphasis> tags. All line
breaks occur outside of the XML tag. This paragraph, of style p first,
is honored by its stylish drop cap.</p> <p>Another paragraph tags along
with a footnote.<fnref></fnref></p> <p>Footnotes can't be tagged
in InDesign, and that's the end of the
story.</p></contentx/ Story>
```

XML content produced by saving an InCopy file as XML

See the Adobe help topics on InCopy, including "Work with stand-alone documents" and "Preparing XML files for K4 or InCopy workflows".

Unfortunately, Adobe is not expansive with explanations about XML and InCopy; you may want to consult the SDK documentation.

Note: For managing InCopy XML files with version control, which some editorial and production workflows require, some content management solutions include InCopy file handling. Search online for InCopy XML workflows.

⁴ A footnote is a special object in InCopy and InDesign; because it is generated as a page component there isn't a way to tag it directly in the InCopy or InDesign. So the footnote is not part of an XML export of an InCopy file.

However, when RTF exported from InCopy is imported into a Word document, the footnote comes along with the RTF content.

Adobe created new interchange document-description formats for InDesign (IDML) and InCopy (ICML) for version CS4. Both of these formats are written in XML format. So technically, they are XML files. However, just as Microsoft did with Word when creating the .docx file format, Adobe has made the IDML (and ICML) file format a "package" of files in an archive format, which is not intuitive for finding what you want to know about XML content within an InDesign file. One of the best resources to start with is "**Adobe InDesign Markup Language (IDML) Cookbook**" which tells you which part of the package describes what XML-related features of the InDesign file. For example, in the XML section of the IDML Cookbook, you find:

"Adjusting a document's structure

A document's structure is stored in the XML/BackingStory.xml file and within the story XML files that are marked up. The BackingStory contains top-level associations between content and structure. Text that is marked up within a story is in the individual story files. *Detailing this relationship is beyond the scope of this document.* The Import XML Template sample demonstrates performing the equivalent of an XML import completely outside InDesign." (italics added)

Notice that even Adobe thinks that this relationship between XML content and Stories (text flows) is hard to explain and "beyond the scope" of their own cookbook.

Note: An InCopy file is a single file, not a package, saved by default with the .icml extension. However, it can contain XML tagged content just like an InDesign file.

To look at an IDML file with XML in it, take any InDesign file in CS4 or CS5 that you have tagged with some XML, and export the file as IDML. To open the .idml archive file, use an archive-unpacking application such as WinZip, and you will see a set of folders and files such as this:

Folders	X	%	Name	Type
[sirinpleNoVrules_table_usirigX5l			Tags.xml	XML Document
Master5preads		⌂	Backiny5tory.xml	XML Document
(13 META-INF				
i4Jl Resources				
P) Spreads				
Stories				

The contents of the files are interrelated. The Tag.xml file determines what is available in the **Tags** palette in InDesign, and how each tag will be color-coded in InDesign.

```

<?xml version="1.0" encoding="UTF-8" standalone="yes"?>

<idPkg:Tags
xmlns:idPkg="http://ns.adobe.com/AdobeInDesign/idml/1.0/packaging"
DOMVersion="7.0">
    <XMLTag Self="XMLTag/a" Name="a">
        <Properties>
            <TagColor type="enumeration">Green</TagColor>
        </Properties>
    </XMLTag>
    <XMLTag Self="XMLTag/b" Name="b">
        <Properties>
            <TagColor type="enumeration">Blue</TagColor>
        </Properties>
    </XMLTag>
    <XMLTag Self="XMLTag/c" Name="c">
        <Properties>
            <TagColor type="enumeration">Yellow</TagColor>
        </Properties>
    </XMLTag>
</idPkg:Tags>

```

The BackingStory is a reference file to track what XML content is located in which Story. It acts as a lookup only; despite its name, it does not contain any Story content.

```

<?xml version="1.0" encoding="UTF-8" standalone="yes"?>

<idPkg:BackingStory
xmlns:idPkg="http://ns.adobe.com/AdobeInDesign/idml/1.0/packaging"
DOMVersion="7.0">
    <XmlStory Self="u5e" AppliedTOCStyle="n" TrackChanges="false"
StoryTitle="$ID/" AppliedNamedGrid="n">
        <ParagraphStyleRange
AppliedParagraphStyle="ParagraphStyle/[No paragraph style]">
            <CharacterStyleRange
AppliedCharacterStyle="CharacterStyle/$ID/[No character style]">
                <XMLElement Self="di2" MarkupTag="XMLTag/Root">
                    <XMLElement Self="di2i3"
MarkupTag="XMLTag/Story" XMLContent="u7 33"/>
                    <XMLElement Self="di2i5"
MarkupTag="XMLTag/Story" XMLContent="ul66e

```

```

                                <XMLElement Self="di2i6"
MarkupTag="XMLTag/Story" XMLContent="ul6fb"/>
                                </XMLElement>
                                <Content></Content>
                                </CharacterStyleRange>
                                </ParagraphStyleRange>
                                </XmlStory>
</idPkg:Backingstory>

```

If you open the u733.xml file in the Stories folder, you will see that it starts off like this:

```

<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<idPkg:Story
    "http://ns.adobe.com/AdobeInDesign/idml/1.0/packaging"
DOMVersion="7.0">
    <Story Self="u733" AppliedTOCStyle="n" TrackChanges="false"
StoryTitle="$ID/" AppliedNamedGrid="n">

```

Then somewhere further down in the file you will see an XML element, with <Content> nested inside it:

```

<XMLElement Self="di2i3ilib" MarkupTag="XMLTag/c"
XMLContent="u733il5e2i8">
    <ParagraphStyleRange
AppliedParagraphStyle="ParagraphStyle/tablecell">
        <CharacterStyleRange
AppliedCharacterStyle="CharacterStyle/$ID/[No character style]">
            <Content>element c</Content>
        </CharacterStyleRange>
    </ParagraphStyleRange>
</XMLElement>

```

So you can see the relationship between the BackingStory.xml, pointing to a Story file identified as "u733", and the Tags.xml file, defining a tag for an element <c>, and in u733.xml, the use of the "c" tag in the MarkupTag attribute of the <XMLElement>.

When used with the InDesign Server and well-designed InDesign document templates, automating IDML provides a way to generate beautiful documents from XML content stored in a database or content management system. The Adobe specification and documentation for IDML runs to hundreds of pages. Most of the people who work with the IDML format of InDesign are XML experts.

No one in her right mind wants to read IDML code. But believe it or not, if you can generate exactly the required code in each type of file in the entire IDML package, and put the files in the correct folder structure, zip them and name them with an .idml extension, you too can create an entire InDesign file from code. You can create your own tags and tag your XML without even owning InDesign. I'll leave you with that thought.

Adobe InDesign is a ton of fun for someone like me who loves XML and publishing. I think that Adobe has given the world some serious tools to work with, and they are likely to keep improving the products.

If you work with InDesign as a layout tool, you'll find that there are lots of ways to add value to your skills by using XML. It's a bit esoteric for most layout people, but if you try a few things, XML tends to get your interest.

You can combine XML content with non-XML content in a single document (and I'm not talking about just headers and footers and boilerplate text). You can have multiple Story elements in a Root element and flow from text frame to text frame, threading frames to jump from page to page as desired. You can use the list numbering and other automatic formatting in InDesign on XML content. And it's easy to style XML content with colors, fonts, and even skew effects in InDesign.

```

V  pology
  ^  metadata
V  HL t4skboiyy
  ? ij. Steps
    T Ustep
      ==> cmd "Retrieve"
    > B <sp
      E> IB step
V  concept
  -J_ title "01TA within InDesigni
V  ctbody
  _j! "It is possible to crea
i^J w Copyright IBM Corp. 10M, 2001

```

Replace the cover of your system.
The cover need* to be put bach ori to
reduce problems from dust.
Keyword: hard drive. Computer cover

6. Retrieve the computer's cover from its safe place. Put it bach on.
7. Retrieve the screws from the safe place. Put them bad? in.
8. Pi it away your screw-driver before you lose it.

#DI?A within ^InDesign Is Fragile

- 9 It is possible to create, edit and appeixf or delete DI7A XML 'content within InDesign. To mini-•inizf mistakes, you liave to switch •between the Stoiv Editor and tfie ^ Lajwut views and constantly dieck (^ that the XML tagging is correct. It isveiyeasytoiwdvertentiytagan *
- element inajrectly in a way tixit •
- wilt make the file invalid. •

Even more enticing is the ability to use a different set of styles and layout to completely change how XML content looks, solely by making new InDesign templates with new style definitions. The InDesign template provides the "skin" for the content, so any XML that maps to the styles can be changed to a different look just by using a different InDesign template.

If you are an XML-whacker already, but you haven't ever used it in a desktop publishing application, I hope that the **Advanced Topics** section has gotten you interested. You are sure to try something that I haven't covered here, so consider letting us know by feeding back your comments to improve this Short Cut.

If you want to see Adobe continue to enhance the XML capabilities for do-it-yourselfers (not just for people who can afford high-end server-based Adobe apps), then send them suggestions and show off what you do with XML and InDesign. Personally, I'm planning to try a lot more types of layouts and XSLT with InDesign, and maybe try my hand at scripting more XML, too.

Resources

The InDesign forum at Adobe's site (<http://www.adobe.com/support/forums/>) is a good place to find examples, pose questions and get help on any XML import, tagging, or export matter.

InDesign Resources

Real World Adobe InDesign CS5 (Peachpit Press) and previous editions for other versions, by Olav Martin Kvern, David Blatner, and Bob Bringhurst.

Adobe InDesign CS4 One-on-One

Adobe InDesign CS3 and XML: A Technical Reference

http://www.adobe.com/products/indesign/scripting/pdfs/indesign_and_xml_technical_reference.pdf (an older reference but it contains a lot of useful content)

Scripting InDesign with JavaScript by Peter Kahrel.

XML Resources

XML in a Nutshell Second Edition

XML Elements of Style

XSLT Resources

XSLT Cookbook, Second Edition *(my favorite XSLT book)*

Acknowledgements

My thanks to the great team I worked with at Monroe Community College: Carol, Bob, Janet, Vince, and Sean. "Make it easy for Vince" was our motto.