

# Designing a New XML Application

---

The last several chapters discussed XML applications that were already written by other people and showed you how to use them. This chapter shows you how to develop an XML application from scratch. In this chapter you see the gradual development of an XML application and associated DTDs for genealogical data.

## Organization of the Data

When developing an XML application from scratch, you need to organize, either in your head or on paper, the data you're describing. There are three basic steps in this process:

1. List the elements.
2. Identify the fundamental elements.
3. Relate the elements.

One of the easiest ways to start the process is to explore the forms and reports that are already available from other formats that describe this data. Genealogy is a fairly well-established discipline, and genealogists have a fairly good idea of what information is and is not useful and how it should be arranged. This is often included in a family group sheet, a sample of which is shown in Figure 23-1.

You'll need to duplicate and organize the fields from the standard reports in your DTD to the extent that they match what you want to do. You can of course supplement or modify them to fit your specific needs.



### In This Chapter

Organization of the data

The person DTD

The family DTD

The source DTD

The family tree

A style sheet for family trees





Figure 23-1: A family group sheet

Note

Object-oriented programmers will note many similarities between what's described in this section and the techniques they gather user requirements. This is partly the result of my own experience and prejudices as an object-oriented programmer, but more of it is due to the similarity of the tasks involved. Gathering user requirements for software is not that different from gathering user requirements for markup languages. Database designers may also notice a lot of similarity between what's done here and what they do when designing a new database.

## Listing the Elements

The first step in developing DTDs for a domain is to decide what the elements are. This isn't hard. It mostly consists of brainstorming to determine what may appear in the domain. As an exercise, write down everything you can think of that may be genealogical information. To keep the problem manageable, include only genealogical data. Assume you can use the XHTML DTD from Chapter 20 for standard text information such as paragraphs, page titles, and so forth. Again, include only elements that specifically apply to genealogy.

Don't be shy. It's easy to remove information later if there's too much or something doesn't prove useful. At this stage, expect to have redundant elements or elements you'll throw away after further thought.

Here's the list I came up with. Your list will almost certainly be at least a little different. And of course you may have used different names for the same things. That's okay. There's no one right answer (which is not to say that all answers are created equal or that some answers aren't better than others).

father	family
parent	birthday
baptism	burial
note	surname
aunt	grandmother
mother	son
child	death date
adoption	grandfather
grave site	given name
niece	uncle
person	daughter
baby	marriage
gender	date
source	middle name
grandparent	nephew

## Identifying the Fundamental Elements

The list in the last section has effective duplicates and some elements that aren't really necessary. It is probably missing a few elements too, which you'll discover as you continue. This is normal. Developing an XML application is an iterative process that takes some time before you feel comfortable with the result.

What you really need to do at this stage is determine the fundamental elements of the domain. These are likely to be those elements that appear as immediate children of the root, rather than contained in some other element. There are two real possibilities here: family and person. Most of the other items in the list are either characteristics of a person or family (occupation, birthday, marriage) or they're a kind of family or person (uncle, parent, baby).

At this stage, most people's instinct is to say that family is the only fundamental element, and that families contain people. This is certainly consistent with the usage of the terms *parent* and *child* to describe the relationships of XML elements (a usage I eschew in this chapter to avoid confusion with the human parents and children we're modeling). For example, you might imagine that a family looks like this:

```
<FAMILY>
  <HUSBAND>Samuel English Anderson</HUSBAND>
  <WIFE>Cora Rucker McDaniel</WIFE>
  <CHILD>Judson McDaniel Anderson</CHILD>
  <CHILD>Thomas Corwin Anderson</CHILD>
  <CHILD>Rodger French Anderson</CHILD>
  <CHILD>Mary English Anderson</CHILD>
</FAMILY>
```

However, there's a problem with this approach. A single person likely belongs to more than one family. I am both the child of my parents and the husband of my wife. That's two different families. Perhaps you can think of this as one extended family, but how far back does this go? Are my grandparents part of the same family? My great-grandparents? My in-laws? Genealogists generally agree that for the purposes of keeping records, a family is a mother, a father, and their children.

Of course, the real world isn't even that simple. Some people have both adoptive and biological parents. Many people have more than one spouse over a lifetime. My father-in-law, Sidney Hart Anderson, was married 15 separate times to 12 different women. Admittedly, Sidney is an extreme case. When he died, he was only four marriages away from tying the world record for serial marriage. (Since then, former Baptist minister Glynn Wolfe pushed the record to 28 consecutive marriages.) Nonetheless, you do need to account for the likelihood that the same people belong to different families.

The standard family group sheets used by the Mormons, a variation of which was shown in Figure 23-1, account for this by repeating the same people and data on different sheets. But for computer applications it's better not to store the same information more than once. Among other things, this avoids problems where data stored in one place is updated while data stored in another is not. Instead, you can make connections between different elements by using `ID` and `IDREF` attributes.

Thus it is not enough to have only a single fundamental family element. There must be at least one other fundamental element, the person. Each person is unique. Each has a single birthday, a single death date, most of the time (though not always) a single name, and various other data. Families are composed of different collections of persons. By defining the persons who make up a family, as well as their roles inside the family, you define the family.

Note

We often think of our family as an extended family including grandparents, daughters-in-law, uncles, aunts, and cousins, and perhaps biologically unrelated individuals who happen to live in the same house. However, in the context of genealogy, a family is a single pair of parents and their children. In some cases the names of these people may be unknown, and in many cases there may be no children or no husband or wife (a single individual qualifies as a family of one). However, a family does not include more distant relationships. A large part of genealogy is the establishment of the actual biological or adoptive relationships between people. It's not uncommon to discover in the course of one's research that the "Cousin Puss" or "Aunt Moot" referred to in old letters was in fact no relation at all! Such people should certainly be included in your records, but failure to keep their actual connections straight can only lead to confusion farther down the road.

There's one more key element that may or may not be a direct child of the root. That's the source for information. A source is like a bibliographical footnote, specifying where each piece of information came from. The source may be a magazine article such as "Blaise Pradel, *Man At Arms*, May/June 1987, p. 26-31"; a book like "*A Sesquicentennial History of Kentucky* by Frederik A. Wallis & Hambleon Tapp, 1945, The Historical Record Association, Hopkinsville, KY"; a family bible such as "English-Demint Anderson Bible, currently held by Beth Anderson in Brooklyn"; or simply word of mouth such as "Anne Sandusky, interview, 6-12-1995".

Tracking the source for a particular datum is important because different sources often disagree. It's not uncommon to see birth and death dates that differ by a day or a year, plus or minus. Less common but still too frequent are confusions between parents and grandparents, aunts and cousins, names of particular people, and more. When you uncover information that disputes information you've already collected, it's important to make a reasonable judgement about whether the new information is more reliable than the old. Not all sources are reliable. In my own research I've found a document claiming to trace my wife's lineage back to Adam and Eve through various English royalty from the Middle Ages, and assorted biblical figures. Needless to say, I don't take this particular source very seriously.

I can think of plausible reasons to make the source a child of the individual elements it documents, but ultimately I think the source is not part of a person or a family in the same way that a birth date or marriage date belongs to a particular person. Rather, it is associated information that should be stored separately and referenced through an ID. The main reason is that a single source such as an old family bible may well contain data about many different elements. In keeping with principles of data normalization, I'd prefer not to repeat the information about the source more than once in the document. If you like, think of this as akin to using end notes rather than footnotes.

## Establishing Relationships Among the Elements

The third and final step before the actual writing of the DTD is to identify how the different pieces of information you want to track are connected. You've determined that the three fundamental elements are the person, the family, and the source. Now you must decide what you want to include in these fundamental elements.

### FAMILY

A family is generally composed of a husband, a wife, and zero or more children. Either the husband or the wife is optional. If you wish to account for same-sex marriages (something most genealogy software couldn't do until recently), simply require one or two parents or spouses without specifying gender. Gender may then be included as an attribute of a person, which is where it probably belongs anyway.

Is there other information associated with a family, as opposed to individuals in the family? I can think of one thing that is important to genealogists: marriage information. The date and place a couple was married (if any) and the date and place a couple was divorced (again, if any), are important information. Although you could include such dates as part of each married individual, it really makes sense to make it part of the family. Given that, a family looks something like this:

```
<FAMILY>
  <MARRIAGE>
    <DATE>...</DATE>
    <PLACE>...</PLACE>
  </MARRIAGE>
  <DIVORCE>
    <DATE>...</DATE>
    <PLACE>...</PLACE>
  </DIVORCE>
  <HUSBAND>...</HUSBAND>
  <WIFE>...</WIFE>
  <CHILD>...</CHILD>
  <CHILD>...</CHILD>
  <CHILD>...</CHILD>
</FAMILY>
```

Information can be omitted if it isn't relevant (such as a couple never divorced) or if you don't have it.

### PERSON

The PERSON element is likely to be more complex. Let's review the standard information you'd want to store about a person:

- ♦ name
- ♦ birth

- ♦ baptism
- ♦ death
- ♦ burial
- ♦ father
- ♦ mother

Of these, name, birth, death, and burial are likely to be elements contained inside a person. Father and mother are likely to be attributes of the person that refer back to the person elements for those people. Furthermore, a person needs an `ID` attribute so he or she can be referred to by family and other person elements.



Father and mother seem to be borderline cases where you might get away with using attributes, but there is the potential to run into trouble. Although everyone has exactly one biological mother and one biological father, many people have adoptive parents that may also need to be connected to the person.

Names are generally divided into family name and given name. This allows you to do things like write a style sheet that boldfaces all people with the last name “Harold”.

Birth, death, burial (and possibly baptism, sometimes a baptismal record is all that’s available for an individual) can all be divided into a date (possibly including a time) and a place. Again, the place may simply be `CDATA` or it can even be a full address element. However, in practice, full street addresses a post office could deliver to are not available. Much more common are partial names like “Mount Sterling, Kentucky” or the name of an old family farm.

Dates can either be stored as `CDATA` or broken up into day, month, and year. In general, it’s easier to break them into day, month, and year than to stick to a common format for dates. On the other hand, allowing arbitrary text inside a date element also allows for imprecise dates like “1919-20”, “before 1753”, or “about 1800”.

That may seem like everything, but we’ve left out one of the most interesting and important pieces of all — notes. A note about a person may contain simple data like “first Eagle Scout in Louisiana” or it may contain a complete story, like how Sam Anderson was killed in the field. This may be personal information like religious affiliation, or it may be medical information like which ancestors died of stomach cancer. If you’ve got a special interest in particular information like religion or medical history, you can make that a separate element of its own, but you should still include some element that can hold arbitrary information of interest that you dig up during your research.

There are other things you could include in a `PERSON` element, photographs for instance, but for now I’ll stop here so this chapter is manageable. Let’s move on to the `SOURCE` element.

## SOURCE

The third and final top-level element is `SOURCE`. A source is bibliographic information that says where you learned a particular fact. It can be a standard citation to a published article or book such as “Collin’s History of Kentucky, Volume II, p. 325, 1840, 1875”. Sources like this have a lot of internal structure that could be captured with elements like `BOOK`, `AUTHOR`, `VOLUME`, `PAGE_RANGE`, `YEAR`, and so forth. Several efforts are currently underway to produce DTDs for generic bibliographies.

However, sources in genealogy tend to be lot messier than in the typical term paper. For instance, one of the most important sources in genealogy can be the family bible with records of births, dates, and marriages. In such a case, it’s not the edition, translation, or the publisher of the bible that’s important; it’s the individual copy that resides in Aunt Doodie’s house. For another example, consider exactly how you cite an obituary you found in a 50-year-old newspaper clipping in a deceased relative’s purse. Chances are the information in the obituary is close to accurate, but it’s not easy to figure out exactly what page of what newspaper on what date it came from.

Since developing an XML application for bibliographies could easily be more than a chapter of its own, and is a task best left to professional librarians, I will satisfy myself with making the `SOURCE` element contain only character data. It will also have an `ID` attribute in the form `s1`, `s2`, `s3`, and so forth, so that each source can be referred to by different elements. Let’s move on to writing the DTD that documents this XML application.

## The Person DTD

By using external entity references, it’s possible to store individual people in separate files, then pull them together into families and family trees later. So let’s begin with a DTD that works for a single person. We’ll merge this into a DTD for families and family trees in the next section.

To develop a DTD, it’s often useful to work backwards — that is, first write out the XML markup you’d like to see using a real example or two, then write the DTD that matches the data. I’m going to use my great-grandfather-in-law Samuel English Anderson as an example, because I have enough information about him to serve as a good example, and also because he’s been dead long enough that no one should get upset over anything I say about him. (You’d be amazed at the scandals and gossip you dig up when doing genealogical research.) Here’s the information I have about Samuel English Anderson:

**Name:** Samuel English Anderson<sup>29, 43</sup>

**Birth:** 25 Aug 1871      Sideview

**Death:** 10 Nov 1919      Mt. Sterling, KY

**Father:** Thomas Corwin Anderson (1845-1889)

**Mother:** LeAnah (Lee Anna, Annie) DeMint English (1843-1898)

### Misc. Notes<sup>219</sup>

Samuel English Anderson was known in Montgomery County for his red hair and the temper that went with it. He did once *kill a man*, but the court found that it was in self defense.

He was shot by a farm worker whom he had fired the day before for smoking in a tobacco barn. Hamp says this may have been self-defense, because he threatened to kill the workers for smoking in the barn. He also claims that old-time rumors say they mashed his head with a fence post. Beth heard he was cut to death with machetes in the field, but Hamp says they wouldn't be cutting tobacco in November, only stripping it in the barn.

Now let's reformat this into XML as shown in Listing 23-1:

### Listing 23-1: An XML document for Samuel English Anderson

```
<?xml version="1.0"?>
<!DOCTYPE PERSON SYSTEM "person.dtd">
<PERSON ID="p37" SEX="M">
  <REFERENCE SOURCE="s29"/>
  <REFERENCE SOURCE="s43"/>
  <NAME>
    <GIVEN>Samuel English</GIVEN>
    <SURNAME>Anderson</SURNAME>
  </NAME>
  <BIRTH>
    <PLACE>Sideview</PLACE>
    <DATE>25 Aug 1871</DATE>
  </BIRTH>
  <DEATH>
    <PLACE>Mt. Sterling, KY</PLACE>
    <DATE>10 Nov 1919</DATE>
  </DEATH>
  <SPOUSE PERSON="p1099"/>
  <SPOUSE PERSON="p2660"/>
  <FATHER PERSON="p1035"/>
  <MOTHER PERSON="p1098"/>
  <NOTE>
    <REFERENCE SOURCE="s219"/>
    <body>
      <p>
        Samuel English Anderson was known in Montgomery County
        for his red hair and the temper that went with it. He
        did once <strong>kill a man</strong>, but the court
        found that it was in self-defense.
      </p>
    </body>
  </NOTE>
</PERSON>
```

*Continued*

## Listing 23-1 (continued)

```

    </p>
    <p>
      He was shot by a farm worker whom he had
      fired the day before for smoking in a tobacco barn.
      Hamp says this may have been self-defense, because he
      threatened to kill the workers for smoking in the barn.
      He also says old-time rumors say they mashed his head
      with a fence post. Beth heard he was cut to death with
      machetes in the field, but Hamp says they wouldn't be
      cutting tobacco in November, only stripping it in the
      barn.
    </p>
  </body>
</NOTE>
</PERSON>

```

The information about other people has been removed and replaced with references to them. The ID numbers are provided by the database I use to store this information (Reunion 5.0 for the Mac from Leister Productions). The end note numbers become SOURCE attributes of REFERENCE elements. HTML-like tags are used to mark up the note.

Now let's see what a DTD for this would look like. The first element is PERSON. This element may contain names, references, births, deaths, burials, baptisms, notes, spouses, fathers, and mothers. I'm going to allow zero or more of each in any order.

```

<!ELEMENT PERSON (NAME | REFERENCE | BIRTH | DEATH | BURIAL
  | BAPTISM | NOTE | SPOUSE | FATHER | MOTHER )*>

```

At first glance it may seem strange not to require a BIRTH or some of the other elements. After all, everybody has exactly one birthday. However, keep in mind that what's being described here is more our knowledge of the person than the person him or herself. You often know about a person without knowing the exact day or even year they were born. Similarly, you may sometimes have conflicting sources that give different values for birthdays or other information. Therefore, it may be necessary to include extra data.

The PERSON element has two attributes, an ID, which we'll require, and a SEX which we'll make optional. (Old records often contain children of unspecified gender, sometimes named, sometimes not. Even photographs don't always reveal gender, especially when children who died very young are involved.)

```

<!ATTLIST PERSON
  ID ID #REQUIRED
  SEX (M | F) #IMPLIED>

```

Next the child elements must be declared. Four of them — BIRTH, DEATH, BURIAL, and BAPTISM—consist of a place and a date, and are otherwise the same. This is a good place for a parameter entity reference:

```
<!ENTITY % event "(REFERENCE*, PLACE?, DATE?)*">
<!ELEMENT BIRTH %event;>
<!ELEMENT BAPTISM %event;>
<!ELEMENT DEATH %event;>
<!ELEMENT BURIAL %event;>
```

I've also added one or more optional REFERENCE element at the start, even though this example doesn't have a SOURCE for any event information. Sometimes, you'll have different sources for different pieces of information about a person. In fact I'll add REFERENCE elements as potential children of almost every element in the DTD. I declare REFERENCE like this, along with a comment in case it isn't obvious from glancing over the DTD exactly what's supposed to be found in the reference:

```
<!-- The ID number of a REFERENCE element
      that documents this entry -->
<!ELEMENT REFERENCE EMPTY>
<!ATTLIST REFERENCE SOURCE NMTOKEN #REQUIRED>
```

Here the SOURCE attribute merely contains the number of the corresponding source. When actual SOURCE elements are added to the DTD below, this can become the ID of the SOURCE element.

A PLACE contains only text. A DATE contains a date string. I decided against requiring a separate year, date, and month to allow for less-certain dates that are common in genealogy like "about 1876" or "sometime before 1920".

```
<!ELEMENT PLACE (#PCDATA)>
<!ELEMENT DATE (#PCDATA)>
```

The SPOUSE, FATHER, and MOTHER attributes each contain a link to the ID of a PERSON element via a PERSON attribute. Again, this is a good opportunity to use a parameter entity reference:

```
<!ENTITY % personref "PERSON NMTOKEN #REQUIRED">
<!ELEMENT SPOUSE EMPTY>
<!ATTLIST SPOUSE %personref;>
<!ELEMENT FATHER EMPTY>
<!ATTLIST FATHER %personref;>
<!ELEMENT MOTHER EMPTY>
<!ATTLIST MOTHER %personref;>
```

Ideally, the PERSON attribute would have type IDREF. However, as long as the person being identified may reside in another file, the best you can do is require a name token type.

The NAME element may contain any number of REFERENCE elements and zero or one SURNAME and GIVEN elements. Each of these may contain text.

```
<!ELEMENT NAME (REFERENCE*, GIVEN?, SURNAME?)>
<!ELEMENT GIVEN (#PCDATA)>
<!ELEMENT SURNAME (#PCDATA)>
The NOTE element may contain an arbitrary amount of text. Some
standard markup would be useful here. The easiest solution is
to adopt the XHTML DTD introduced in Chapter 20, Reading
Document Type Definitions. It's not necessary to rewrite it
all. Simply use a parameter reference entity to import it.
We'll allow each NOTE to contain zero or more REFERENCE
elements and a single body element.<!ENTITY % xhtml SYSTEM
"xhtml/XHTML1-s.dtd">
%xhtml;
<!ELEMENT NOTE (REFERENCE*, body)>
```

Those three little lines get you the entire HTML 4.0 markup set. There's no need to invent your own. You can use the already familiar and well-supported HTML tag set. I have left out the header, though that would be easy to include — just by replacing body with html in the above. (I left it out because doing so also requires you to include head and title elements, which seemed superfluous here.) This does assume that the file XHTML1-s.dtd can be found at the relative URL xhtml/XHTML1-s.dtd, though that's easy to adjust if you want to put it somewhere else. You could even use the absolute URL at the W3C Web site, <http://www.w3.org/TR/xhtml-modularization/DTD/XHTML1-s.dtd>, though I prefer not to make my files dependent on the availability of a Web site I don't control. Listing 23-2 shows the complete DTD.

### Listing 23-2: person.dtd: The complete PERSON DTD

```
<!ELEMENT PERSON ( NAME | REFERENCE | BIRTH | DEATH | BURIAL
| BAPTISM | NOTE | FATHER | MOTHER | SPOUSE )* >
<!ATTLIST PERSON ID ID #REQUIRED>

<!--M means male, F means female -->
<!ATTLIST PERSON SEX ( M | F ) #IMPLIED>

<!-- The ID number of a SOURCE element that documents
this entry -->
<!ELEMENT REFERENCE EMPTY>
<!ENTITY % sourceref "SOURCE NMTOKEN #REQUIRED">
<!ATTLIST REFERENCE %sourceref;>

<!ENTITY % event "(REFERENCE*, PLACE?, DATE?)">
<!ELEMENT BIRTH %event;>
<!ELEMENT BAPTISM %event;>
<!ELEMENT DEATH %event;>
```

```

<!ELEMENT BURIAL %event;>

<!ELEMENT PLACE (#PCDATA)>
<!ELEMENT DATE (#PCDATA)>

<!ENTITY % personref "PERSON NMTOKEN #REQUIRED">
<!ELEMENT SPOUSE EMPTY>
<!ATTLIST SPOUSE %personref;>
<!ELEMENT FATHER EMPTY>
<!ATTLIST FATHER %personref;>
<!ELEMENT MOTHER EMPTY>
<!ATTLIST MOTHER %personref;>

<!ELEMENT NAME (GIVEN?, SURNAME?)>
<!ELEMENT GIVEN (#PCDATA)>
<!ELEMENT SURNAME (#PCDATA)>

<!ENTITY % xhtml SYSTEM "xhtml/XHTML1-s.dtd">
%xhtml;
<!ELEMENT NOTE (REFERENCE*, body)>

```

---

## The Family DTD

The next step is to write a DTD for a family. Let's begin with a sample family XML document, as shown in Listing 23-3:

### Listing 23-3: An XML document for Samuel English Anderson's family

```

<?xml version="1.0" standalone="no"?>
<!DOCTYPE FAMILY SYSTEM "family.dtd">
<FAMILY>
  <HUSBAND PERSON="p37"/>
  <WIFE PERSON="p1099"/>
  <CHILD PERSON="p23"/>
  <CHILD PERSON="p36"/>
  <CHILD PERSON="p1033"/>
  <CHILD PERSON="p1034"/>
  <MARRIAGE>
    <PLACE>Cincinnati, OH</PLACE>
    <DATE>15 Jul 1892</DATE>
  </MARRIAGE>
</FAMILY>

```

---

All that's needed here are references to the members of the family, not the actual family members themselves. The reference `PERSON` IDs are once again provided from the database where this information is stored. Their exact values aren't important as long as they're reliably unique and stable.

Now that you've got a sample family, you have to prepare the DTD for all families, like the one shown in Listing 23-4. Don't forget to include items that are needed for some families — even if not for this example — like a divorce. A parameter entity reference will pull in the declarations from the person DTD of Listing 23-2.

#### Listing 23-4: `family.dtd`: A DTD that describes a family

```
<!ENTITY % person SYSTEM "person.dtd">
%person;

<!ELEMENT FAMILY (REFERENCE*, HUSBAND?, WIFE?, CHILD*,
                  MARRIAGE*, DIVORCE*, NOTE*)>

<!ELEMENT HUSBAND EMPTY>
<!ATTLIST HUSBAND %personref;>
<!ELEMENT WIFE EMPTY>
<!ATTLIST WIFE %personref;>
<!ELEMENT CHILD EMPTY>
<!ATTLIST CHILD %personref;>
<!ELEMENT DIVORCE %event;>
<!ELEMENT MARRIAGE %event;>
```

I'm assuming no more than one `HUSBAND` or `WIFE` per `FAMILY` element. This is a fairly standard assumption in genealogy, even in cultures where plural marriages are common, because it helps to keep the children sorted out. When documenting genealogy in a polygamous society, the same `HUSBAND` may appear in multiple `FAMILY` elements. When documenting genealogy in a polyandrous society, the same `WIFE` may appear in multiple `FAMILY` elements. Aside from overlapping dates, this is essentially the same procedure that's followed when documenting serial marriages. And of course there's nothing in the DTD that actually requires people to be married in order to have children (any more than there is anything in biology that requires it).

Overall, this scheme is very flexible, much more so than if a `FAMILY` element had to contain individual `PERSON` elements rather than merely pointers to them. That would almost certainly require duplication of data across many different elements and files. The only thing this DTD doesn't handle well are same-sex marriages, and that could easily be fixed by changing the `FAMILY` declaration to the following:

```
<!ELEMENT FAMILY (((HUSBAND, WIFE) | (HUSBAND, HUSBAND?)
| (WIFE, WIFE?)), MARRIAGE*, DIVORCE*, CHILD*)>
```

Allowing multiple marriages and divorces in a single family may seem a little strange, but it does happen. My mother-in-law married and divorced my father-in-law three separate times. Remarriages to the same person aren't common, but they do happen.

## The Source DTD

The third and final top-level element is `SOURCE`. I'm using a watered-down `SOURCE` element with little internal structure. However, by storing the DTD in a separate file, it would be easy to add structure to it later. Some typical `SOURCE` elements look like this:

```
<SOURCE ID="s218">Hamp Hoskins interview, 11-28-1996</SOURCE>
<SOURCE ID="s29">English-Demint Anderson Bible</SOURCE>
<SOURCE ID="s43">Anderson Bible</SOURCE>
<SOURCE ID="s43">
  Letter from R. Foster Adams to Beth Anderson, 1972
</SOURCE>
<SOURCE ID="s66">
  Collin's History of Kentucky, Volume II, p.325, 1840, 1875
</SOURCE>
```

A `SOURCE` element probably has a lot of internal structure. Work is ongoing in several places to produce a generic DTD for bibliographic information with elements for articles, authors, pages, publication dates, and more. However, this is quite a complex topic when considered in its full generality, and as previously mentioned, it doesn't work quite the same for genealogy as it does for most fields. The individual copy of a family bible or newspaper clipping with handwritten annotations may be more significant than the more generic, standard author, title, publisher data used in most bibliographies.

Because developing an XML application for bibliographies could easily be more than a chapter of its own, and is a task best left to professional librarians, I will satisfy myself with making the `SOURCE` element contain only character data. It will also have an `ID` attribute in the form `s1`, `s2`, `s3`, and so forth, so that each source can be referred to by different elements. Listing 23-5 shows the extremely simple DTD for sources.

### Listing 23-5: source.dtd: A simple SOURCE DTD

```
<!ELEMENT SOURCE (#PCDATA)>
<!ATTLIST SOURCE ID ID #REQUIRED>
```

## The Family Tree DTD

It's now possible to combine the various families, people, and sources into a single grouping that includes everyone. I'll call the root element of this document `FAMILY_TREE`. It will include `PERSON`, `FAMILY` and `SOURCE` elements in no particular order:

```
<!ELEMENT FAMILY_TREE (PERSON | FAMILY | SOURCE)*>
```

It's not necessary to re-declare the `PERSON`, `FAMILY` and `SOURCE` elements and their children. Instead, these can be imported by importing the family and source DTDs with external parameter entity references. The family DTD then imports the person DTD:

```
<!ENTITY % family SYSTEM "family.dtd">
%family;
<!ENTITY % source SYSTEM "source.dtd">
%source;
```

One thing you want to do at this point is switch from using `NMTOKEN` types for spouses, parents, and references to actual `ID` types. This is because a `FAMILY` element that's part of a `FAMILY_TREE` should include all necessary `PERSON` elements. You can do that by overriding the `personref` and `sourceref` parameter entity declarations in the DTD for the family tree:

```
<!ENTITY % personref "PERSON IDREF #REQUIRED">
<!ENTITY % sourceref "SOURCE IDREF #REQUIRED">
```

That's all you need. Everything else is contained in the imported person and family DTDs. Listing 23-6 shows the family-tree DTD. Listing 23-7 shows a complete family tree document that includes 11 people, three families, and seven sources.

### Listing 23-6: familytree.dtd: The family-tree DTD

```
<!ENTITY % personref "PERSON IDREF #REQUIRED">
<!ENTITY % sourceref "SOURCE IDREF #REQUIRED">

<!ENTITY % family SYSTEM "family.dtd">
%family;

<!ENTITY % source SYSTEM "source.dtd">
%source;

<!ELEMENT FAMILY_TREE (SOURCE | PERSON | FAMILY )*>
```

## Listing 23-7: An XML document of a complete family tree

```

<?xml version="1.0" standalone="no"?>
<!DOCTYPE FAMILY_TREE SYSTEM "familytree.dtd">
<FAMILY_TREE>

  <PERSON ID="p23" SEX="M">
    <REFERENCE_SOURCE="s44"/>
    <FATHER_PERSON="p37"/>
    <MOTHER_PERSON="p1099"/>
    <NAME>
      <GIVEN>Judson McDaniel</GIVEN>
      <SURNAME>Anderson</SURNAME>
    </NAME>
    <BIRTH>
      <PLACE>Montgomery County, KY, 1893</PLACE>
      <DATE>19 Jul 1894</DATE>
    </BIRTH>
    <DEATH>
      <PLACE>Mt. Sterling, KY</PLACE>
      <DATE>27 Apr 1941</DATE>
    </DEATH>
    <NOTE><body>
      <p>Agriculture College in Iowa</p>
      <p>Farmer</p>
      <p>32nd degree Mason</p>
      <p>
        He shot himself in the pond in the back of Sideview
        when he found that he was terminally ill. It has also
        been claimed that he was having money and wife
        troubles. (He and Zelda did not get along and he was
        embarrassed to have married her.) It has further been
        claimed that this was part of the Anderson family
        curse.
      </p>
    </body></NOTE>
  </PERSON>

  <PERSON ID="p36" SEX="F">
    <REFERENCE_SOURCE="s43"/>
    <FATHER_PERSON="p37"/>
    <MOTHER_PERSON="p1099"/>
    <NAME>
      <GIVEN>Mary English</GIVEN>
      <SURNAME>Anderson</SURNAME>
    </NAME>
    <BIRTH>
      <PLACE>August 4, 1902?, Sideview, KY</PLACE>
      <DATE>8 Apr 1902</DATE>
    </BIRTH>
    <DEATH>

```

*Continued*

## Listing 23-7 (continued)

```

        <PLACE>Mt. Sterling, KY</PLACE>
        <DATE>19 Dec 1972</DATE>
    </DEATH>
</PERSON>

<PERSON ID="p37" SEX="M">
    <REFERENCE SOURCE="s29"/>
    <REFERENCE SOURCE="s43"/>
    <FATHER PERSON="p1035"/>
    <MOTHER PERSON="p1098"/>
    <NAME>
        <GIVEN>Samuel English</GIVEN>
        <SURNAME>Anderson</SURNAME>
    </NAME>
    <BIRTH>
        <PLACE>Sideview</PLACE>
        <DATE>25 Aug 1871</DATE>
    </BIRTH>
    <DEATH>
        <PLACE>Mt. Sterling, KY</PLACE>
        <DATE>10 Nov 1919</DATE>
    </DEATH>
    <NOTE>
        <body>
            <p>
                Samuel English Anderson was known in Montgomery
                County for his red hair and the temper that went
                with it. He did once <strong>kill a man</strong>,
                but the court found that it was in self-defense.
            </p>

            <p>
                He was shot by a farm worker whom he had
                fired the day before for smoking in a tobacco barn.
                Hamp says this may have been self-defense, because he
                threatened to kill the workers for smoking in the
                barn. He also says old-time rumors say they mashed
                his head with a fence post. Beth heard he was cut to
                death with machetes in the field, but Hamp says they
                wouldn't be cutting tobacco in November, only
                stripping it in the barn.
            </p>
        </body>
    </NOTE>
</PERSON>

<PERSON ID="p1033" SEX="M">
    <REFERENCE SOURCE="s43"/>
    <FATHER PERSON="p37"/>
    <MOTHER PERSON="p1099"/>

```

```

<NAME>
  <GIVEN>Thomas Corwin</GIVEN>
  <SURNAME>Anderson</SURNAME>
</NAME>
<BIRTH>
  <DATE>16 Jan 1898</DATE>
</BIRTH>
<DEATH>
  <PLACE>Probably Australia</PLACE>
</DEATH>
<NOTE>
  <body><p>
    Corwin fought with his father and then left home.
    His last letter was from Australia.
  </p></body>
</NOTE>
</PERSON>

<PERSON ID="p1034" SEX="M">
  <REFERENCE SOURCE="s43"/>
  <FATHER PERSON="p37"/>
  <MOTHER PERSON="p1099"/>
  <NAME>
    <GIVEN>Rodger French</GIVEN>
    <SURNAME>Anderson</SURNAME>
  </NAME>
  <BIRTH>
    <DATE>26 Nov 1899</DATE>
  </BIRTH>
  <DEATH>
    <PLACE>Birmingham, AL</PLACE>
  </DEATH>
  <NOTE>
    <body><p>
      Killed when the car he was driving hit a pig in the
      road; Despite the many suicides in the family, this is
      the only known sowicide.
    </p></body>
  </NOTE>
</PERSON>

<PERSON ID="p1035" SEX="M">
  <NAME>
    <GIVEN>Thomas Corwin</GIVEN>
    <SURNAME>Anderson</SURNAME>
  </NAME>
  <BIRTH>
    <DATE>24 Aug 1845</DATE>
  </BIRTH>
  <DEATH>
    <PLACE>Mt. Sterling, KY</PLACE>
    <DATE>18 Sep 1889</DATE>

```

*Continued*

## Listing 23-7 (continued)

```

</DEATH>
<NOTE>
  <body>
    <p>Yale 1869 (did not graduate)</p>
    <p>Breeder of short horn cattle</p>
    <p>He was named after an Ohio senator. The name Corwin
      is from the Latin <i>corvinus</i> which means
      <i>raven</i> and is akin to <i>corbin</i>/<i>corbet</i>.
      In old French it was <i>cord</i> and in Middle English
      <i>Corse</i> which meant <i>raven</i> or <i>cow</i>.
    </p>
    <p>Attended Annapolis for one year, possibly to
      avoid service in the Civil War.</p>
    <p>He farmed the old Mitchell farm
      and became known as a leading short horn breeder.
      He suffered from asthma and wanted to move to
      Colorado in 1876 to avoid the Kentucky weather, but
      he didn't.
    </p>
  </body>
</NOTE>
</PERSON>

<PERSON ID="p1098" SEX="F">
  <REFERENCE SOURCE="s29"/>
  <NAME>
    <GIVEN>LeAnah (Lee Anna, Annie) DeMint</GIVEN>
    <SURNAME>English</SURNAME>
  </NAME>
  <BIRTH>
    <PLACE>Louisville, KY</PLACE>
    <DATE>1 Mar 1843</DATE>
  </BIRTH>
  <DEATH>
    <REFERENCE SOURCE="s16"/>
    <PLACE>acute Bright's disease, 504 E. Broadway</PLACE>
    <DATE>31 Oct 1898</DATE>
  </DEATH>
  <NOTE>
    <body>
      <p>Writer (pseudonymously) for Louisville Herald</p>
      <p>Ann or Annie was from Louisville. She wrote under
        an assumed name for the Louisville Herald.</p>
    </body>
  </NOTE>
</PERSON>

<PERSON ID="p1099" SEX="F">
  <REFERENCE SOURCE="s39"/>
  <FATHER PERSON="p1100"/>
  <MOTHER PERSON="p1101"/>

```

```
<NAME>
  <GIVEN>Cora Rucker (Blevins?)</GIVEN>
  <SURNAME>McDaniel</SURNAME>
</NAME>
<BIRTH>
  <DATE>1 Aug 1873</DATE>
</BIRTH>
<DEATH>
  <REFERENCE SOURCE="s41"/>
  <REFERENCE SOURCE="s60"/>
  <PLACE>Sideview, bronchial trouble TB</PLACE>
  <DATE>21 Jul 1909</DATE>
</DEATH>
<NOTE>
  <body>
    <p>She was engaged to General Hood of the Confederacy,
      but she was seeing Mr. Anderson on the side. A servant
      was posted to keep Mr. Anderson away. However the girl
      fell asleep, and Cora eloped with Mr. Anderson.</p>
  </body>
</NOTE>
</PERSON>

<PERSON ID="p1100" SEX="M">
  <NAME>
    <GIVEN>Judson</GIVEN>
    <SURNAME>McDaniel</SURNAME>
  </NAME>
  <BIRTH>
    <DATE>21 Feb 1834</DATE>
  </BIRTH>
  <DEATH>
    <DATE>9 Dec 1905</DATE>
  </DEATH>
</PERSON>

<PERSON ID="p1101" SEX="F">
  <NAME>
    <GIVEN>Mary E.</GIVEN>
    <SURNAME>Blevins</SURNAME>
  </NAME>
  <BIRTH>
    <DATE>1847</DATE>
  </BIRTH>
  <DEATH>
    <DATE>1886</DATE>
  </DEATH>
  <BURIAL>
    <PLACE>Machpelah Cemetery, Mt. Sterling KY</PLACE>
  </BURIAL>
</PERSON>
```

*Continued*

## Listing 23-7 (continued)

```

<PERSON ID="p1102" SEX="M">
  <REFERENCE SOURCE="s29"/>
  <NAME>
    <GIVEN>John Jay (Robin Adair )</GIVEN>
    <SURNAME>Anderson</SURNAME>
  </NAME>
  <BIRTH>
    <REFERENCE SOURCE="s43"/>
    <PLACE>Sideview</PLACE>
    <DATE>13 May 1873</DATE>
  </BIRTH>
  <DEATH>
    <DATE>18 Sep 1889    </DATE>
  </DEATH>
  <NOTE><body><p>
    Died of flux. Rumored to have been killed by his brother.
  </p></body></NOTE>
</PERSON>

<FAMILY ID="f25">
  <HUSBAND PERSON="p37"/>
  <WIFE PERSON="p1099"/>
  <CHILD PERSON="p23"/>
  <CHILD PERSON="p36"/>
  <CHILD PERSON="p1033"/>
  <CHILD PERSON="p1034"/>
</FAMILY>

<FAMILY ID="f732">
  <HUSBAND PERSON="p1035"/>
  <WIFE PERSON="p1098"/>
  <CHILD PERSON="p1102"/>
  <CHILD PERSON="p37"/>
</FAMILY>

<FAMILY ID="f779">
  <HUSBAND PERSON="p1102"/>
</FAMILY>

<SOURCE ID="s16">newspaper death notice in purse</SOURCE>
<SOURCE ID="s29">English-Demint Anderson Bible</SOURCE>
<SOURCE ID="s39">
  Judson McDaniel & Mary E. Blevins Bible
</SOURCE>
<SOURCE ID="s41">
  Cora McDaniel obituary, clipping from unknown newspaper
</SOURCE>
<SOURCE ID="s43">Anderson Bible</SOURCE>

```

```

<SOURCE ID="s44">
  A Sesquicentennial History of Kentucky
  Frederik A. Wallis & Hambleon Tapp, 1945,
  The Historical Record Association, Hopkinsville, KY
</SOURCE>
<SOURCE ID="s60">
  Interview with Ann Sandusky, May 1996
</SOURCE>

</FAMILY_TREE>

```

---

## Designing a Style Sheet for Family Trees

The family tree document is organized as a data file rather than a narrative. To get a reasonably pleasing view of the document, you're going to need to reorder and reorganize the contents before displaying them. CSS really isn't powerful enough for this task. Consequently, an XSL style sheet is called for.

It's best to begin with the root node. Here the root node is merely replaced by the standard `html`, `head` and `body` elements. Templates are applied to the `FAMILY_TREE` root element to continue processing.

```

<xsl:template match="/">
  <html>
    <head>
      <title>Family Tree</title>
    </head>
    <body>
      <xsl:apply-templates select="FAMILY_TREE"/>
    </body>
  </html>
</xsl:template>

```

The template rule for the `FAMILY_TREE` element divides the document into three parts, one each for the families, people, and sources. Templates are applied to each separately:

```

<xsl:template match="FAMILY_TREE">

  <h1>Family Tree</h1>

  <h2>Families</h2>
  <xsl:apply-templates select="FAMILY"/>

  <h2>People</h2>
  <xsl:apply-templates select="PERSON"/>

  <h2>Sources</h2>

```

```

    <ul>
      <xsl:apply-templates select="SOURCE"/>
    </ul>

  </xsl:template>

```

**The SOURCE rule is quite simple. Each source is wrapped in a `li` element. Furthermore, its ID is attached using the `name` attribute of the HTML `a` element. This allows for cross-references directly to the source, as shown below:**

```

<xsl:template match="SOURCE">
  <li>
    <xsl:element name="a">
      <xsl:attribute name="name">
        <xsl:value-of select="@ID"/>
      </xsl:attribute>
      <xsl:value-of select="."/>
    </xsl:element>
  </li>
</xsl:template>

```

**The PERSON element is much more complex so we'll break it up into several template rules. The PERSON template rule selects the individual parts, and formats those that aren't too complex. It applies templates to the rest. The name is placed in an `h3` header. This is surrounded with an HTML anchor whose `name` is the person's ID. The BIRTH, DEATH, BAPTISM, and BURIAL elements are formatted as list items, as demonstrated below:**

```

<xsl:template match="PERSON">
  <h3>
    <xsl:element name="a">
      <xsl:attribute name="name">
        <xsl:value-of select="@ID"/>
      </xsl:attribute>
      <xsl:value-of select="NAME"/>
    </xsl:element>
  </h3>

  <ul>
    <xsl:if test="BIRTH">
      <li>Born: <xsl:value-of select="BIRTH"/></li>
    </xsl:if>
    <xsl:if test="DEATH">
      <li>Died: <xsl:value-of select="DEATH"/></li>
    </xsl:if>
    <xsl:if test="BAPTISM">

```

```

        <li>Baptism: <xsl:value-of select="BAPTISM"/></li>
    </xsl:if>
    <xsl:if test="BURIAL">
        <li>Burial: <xsl:value-of select="BURIAL"/></li>
    </xsl:if>
    <xsl:apply-templates select="FATHER"/>
    <xsl:apply-templates select="MOTHER"/>
</ul>

<p>
    <xsl:apply-templates select="NOTE"/>
</p>

</xsl:template>

```

**The FATHER and MOTHER elements are also list items, but they need to be linked to their respective people. These two template rules do that:**

```

<xsl:template match="FATHER">
    <li>
        <xsl:element name="a">
            <xsl:attribute name="href">
                #<xsl:value-of select="@PERSON"/>
            </xsl:attribute>
            Father
        </xsl:element>
    </li>
</xsl:template>

<xsl:template match="MOTHER">
    <li>
        <xsl:element name="a">
            <xsl:attribute name="href">
                #<xsl:value-of select="@PERSON"/>
            </xsl:attribute>
            Mother
        </xsl:element>
    </li>
</xsl:template>

```

**The final thing you need to do to format PERSON elements is to copy the contents of the NOTE into the finished document. Since the body of the NOTE uses standard HTML tags that don't need to be changed, an xsl:copy element is useful. The first of these rules copies the body element itself and its contents:**

```

<xsl:template match="body | body//*">
    <xsl:copy>
        <xsl:apply-templates select="*|@*|comment()|pi()|text()"/>
    </xsl:copy>
</xsl:template>

```

The template rule for FAMILY elements will list the name and role of each member of the family as a list item in an unordered list. Each member will be linked to the description of that individual. The rules to do this look like the following:

```
<xsl:template match="FAMILY">
  <ul>
    <xsl:apply-templates select="HUSBAND"/>
    <xsl:apply-templates select="WIFE"/>
    <xsl:apply-templates select="CHILD"/>
  </ul>
</xsl:template>

<xsl:template match="HUSBAND">
  <li>Husband: <a href="#{@PERSON}">
    <xsl:value-of select="id(@PERSON)/NAME"/>
  </a></li>
</xsl:template>

<xsl:template match="WIFE">
  <li>Wife: <a href="#{@PERSON}">
    <xsl:value-of select="id(@PERSON)/NAME"/>
  </a></li>
</xsl:template>

<xsl:template match="CHILD">
  <li>Child: <a href="#{@PERSON}">
    <xsl:value-of select="id(@PERSON)/NAME"/>
  </a></li>
</xsl:template>
```

The trickiest thing about these rules is the insertion of data from one element (the PERSON) in a template for different elements (HUSBAND, WIFE, CHILD). The ID of the PERSON stored in the HUSBAND/WIFE/CHILD's PERSON attribute is used to locate the right PERSON element; then its NAME child is selected.

Listing 23-8 is the finished family tree style sheet. Figure 23-2 shows the document after it's been converted into HTML and loaded into Netscape Navigator.

### Listing 23-8: The complete family tree style sheet

```
<?xml version="1.0"?>
<xsl:stylesheet
xmlns:xsl="http://www.w3.org/XSL/Transform/1.0">

  <xsl:template match="/">
    <html>
      <head>
        <title>Family Tree</title>
      </head>
      <body>
```

```

        <xsl:apply-templates select="FAMILY_TREE"/>
    </body>
</html>
</xsl:template>

<xsl:template match="FAMILY_TREE">
    <h1>Family Tree</h1>

    <h2>Families</h2>
    <xsl:apply-templates select="FAMILY"/>

    <h2>People</h2>
    <xsl:apply-templates select="PERSON"/>

    <h2>Sources</h2>
    <ul>
        <xsl:apply-templates select="SOURCE"/>
    </ul>

</xsl:template>

<xsl:template match="PERSON">
    <h3>
        <xsl:element name="a">
            <xsl:attribute name="name">
                <xsl:value-of select="@ID"/>
            </xsl:attribute>
            <xsl:value-of select="NAME"/>
        </xsl:element>
    </h3>

    <ul>
        <xsl:if test="BIRTH">
            <li>Born: <xsl:value-of select="BIRTH"/></li>
        </xsl:if>
        <xsl:if test="DEATH">
            <li>Died: <xsl:value-of select="DEATH"/></li>
        </xsl:if>
        <xsl:if test="BAPTISM">
            <li>Baptism: <xsl:value-of select="BAPTISM"/></li>
        </xsl:if>
        <xsl:if test="BURIAL">
            <li>Burial: <xsl:value-of select="BURIAL"/></li>
        </xsl:if>
        <xsl:apply-templates select="FATHER"/>
        <xsl:apply-templates select="MOTHER"/>
    </ul>

    <p>
        <xsl:apply-templates select="NOTE"/>
    </p>

```

*Continued*

## Listing 23-8 (continued)

```

    </p>

</xsl:template>

<xsl:template match="FATHER">
  <li>
    <xsl:element name="a">
      <xsl:attribute name="href">
        #<xsl:value-of select="@PERSON"/>
      </xsl:attribute>
      Father
    </xsl:element>
  </li>
</xsl:template>

<xsl:template match="MOTHER">
  <li>
    <xsl:element name="a">
      <xsl:attribute name="href">
        #<xsl:value-of select="@PERSON"/>
      </xsl:attribute>
      Mother
    </xsl:element>
  </li>
</xsl:template>

<xsl:template match="body | body/**">
  <xsl:copy>
    <xsl:apply-templates
      select="*|@*|comment()|pi()|text()"/>
  </xsl:copy>
</xsl:template>

<xsl:template match="SOURCE">
  <li>
    <xsl:element name="a">
      <xsl:attribute name="name">
        <xsl:value-of select="@ID"/>
      </xsl:attribute>
      <xsl:value-of select="."/>
    </xsl:element>
  </li>
</xsl:template>

<xsl:template match="FAMILY">
  <ul>
    <xsl:apply-templates select="HUSBAND"/>
    <xsl:apply-templates select="WIFE"/>
    <xsl:apply-templates select="CHILD"/>
  </ul>
</xsl:template>

```

```

    </ul>
</xsl:template>

<xsl:template match="HUSBAND">
  <li>Husband: <a href="#{@PERSON}">
    <xsl:value-of select="id(@PERSON)/NAME"/>
  </a></li>
</xsl:template>

<xsl:template match="WIFE">
  <li>Wife: <a href="#{@PERSON}">
    <xsl:value-of select="id(@PERSON)/NAME"/>
  </a></li>
</xsl:template>

<xsl:template match="CHILD">
  <li>Child: <a href="#{@PERSON}">
    <xsl:value-of select="id(@PERSON)/NAME"/>
  </a></li>
</xsl:template>

</xsl:stylesheet>

```

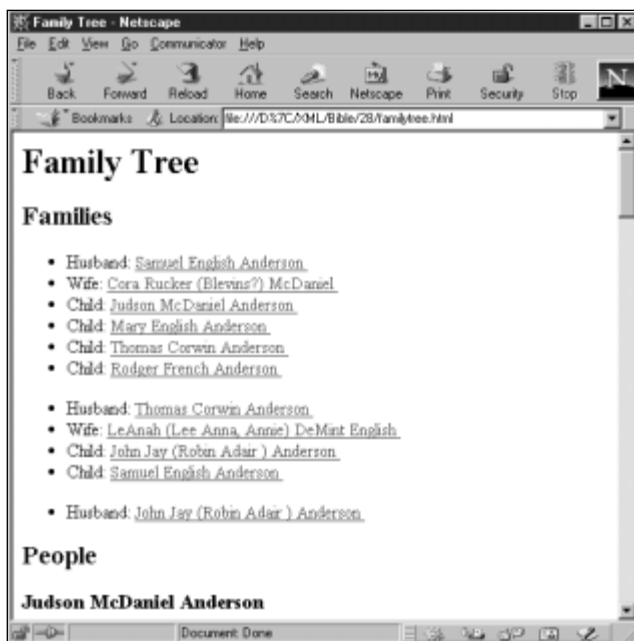


Figure 23-2: The family tree after conversion to HTML

## Summary

In this chapter, you saw an XML application for genealogy developed from scratch. Along the way you have learned:

- ♦ Always begin a new XML application by considering the domain you're describing.
- ♦ Try to identify the fundamental elements of the domain. Everything else is likely to either be contained in or be an attribute of one of these.
- ♦ Try to avoid including the same data in more than one place. Use `ID` and `IDREF` attributes to establish pointers from one element to another.
- ♦ Be sure to consider special cases. Don't base your entire design on the most obvious cases.
- ♦ Use parameter entities to merge the DTDs for each piece of the XML application into one complete DTD.

This concludes the main body of *XML Bible*. Go forth and write your own XML applications! The next several parts provide a variety of useful reference information and the official XML 1.0 Specification.

