1

XML and the Enterprise

XML is short for Extensible Markup Language (sometimes written as eXtensible Markup Language), which enables information to be encoded with meaningful structure and in a way that both computers and humans can understand. It is excellent for information exchange, and is easily extended to include user-specified and industry-specified tags. XML's recommendation — its specifications — is set by the W3C (World Wide Web Consortium).

Because XML is formatting-free, it can be used in a variety of processes. It can replace or work with other technologies, and it can be used instead of or to supplement scripts. It also works with databases or on its own to store readable content.

In this chapter, you:

- Learn the basics of XML.
- Explore the structure of an XML document.
- Discover what you can do with XML.
- Find out what you need to get started.

You are introduced to the winery and will examine the potential for modifying its data and using XML enterprise-wide. This project is expanded upon throughout the book.

Problem

You are owner of a winery in the Finger Lakes area of New York. You've just purchased the winery and are interested in automating much of the administrative detail and marketing data for your newly hired staff to work with.

Your winery's data must be properly structured to enable internal use as well as the sharing of the wine catalog externally. Interoperability with partners, online wine distributors, and tourism agencies is of key importance.

The winery starts with some initial data on the wines that it produces, as well as data on competing wineries. This data must be carefully reviewed and augmented so that by the end of the implementation — the end of this book — the data structure is refined.

The winery has data requirements that run enterprise-wide. The winery needs to have information on its own wines primarily available as a retrievable and accurate list. The inventory control system must be capable of accessing the list and marking, via a Web form, what product is on hand each year. Marketing must pull data from this same set of information to create handouts on the available wines. Up-to-date information on all the winery products and order data should be accessible.

Design

As a text-based format, XML can stand in for text or scripts within programming and scripting files. It also can be used as an authoring markup for producing content. In the case of the winery, there is the potential for using XML information throughout, from storing winery data in a database to drawing XML from the database to use on marketing sheets to outputting sortable information on customer shipments. Before you begin using XML, though, take a quick look at its history.

A Brief History of XML

There are earlier variations of markup languages, but this historic review starts with SGML, the Standard Generalized Markup Language.

SGML and XML's Evolution

SGML was adopted as an international standard (ISO 8879) in 1986. Tools and processes for working with SGML were developed, though not adopted as widely as was hoped.

A markup language like SGML is a text language that enables you to describe your content in a way that is independent of hardware, software, formats, or operating system. Markup language documents are designed to be interchangeable between tools and systems. SGML paved the way for its scaled-down descendant, XML.

The SGML standard, and now the XML standard, enables companies to take information such as the following:

```
Melvin Winery "Muscadine Scupp" Wine,
Vintage 1998
Available in 750ml only.
Our complex purple Muscadine grape lends an assertive aroma and flavor, with a
semi-sweet velvety smooth finish.
```

and put markup information around the data so that it is more specifically identified. To mark up content, you simply surround each element with descriptive tags, most often with a beginning tag and an end tag. Here's the syntax:

```
<element>content</element>
```

where `<element>` represents the beginning of the markup. Here, `element` is a placeholder for whatever your markup element's name right be. Then `</element>` represents where the markup ends, after the content.

For example, the markup of the preceding information looks like this:

```
<wine id="Muscadine_750ml">
<name>Muscadine</name>
<varietal>Scupp</varietal>
<vintage>1998</vintage>
<winery>Melvin</winery>
<bottlesize>750ml</bottlesize>
<description>Our complex purple Muscadine grape lends an assertive aroma and ↩
flavor, with a semi-sweet velvety smooth finish.</description>
</wine>
```

If you are not familiar with the angle brackets, slashes, and other odd pieces of markup in this example, do not worry; all will be explained in detail and soon.

Design of XML

In the mid-1990s, SGML's capability to make sets of markup elements was used to create HTML (Hypertext Markup Language), which provided many with the capability to create markup and display it using browser technologies. But HTML could only mark data as headings or lists or paragraphs or other simple format-oriented content. It could not semantically describe what the content was. Something more powerful was needed.

XML was designed with a narrower scope than SGML so that lighter-weight processors and parsers could be utilized to serve up content via the Internet (and other venues).

The W3C's "XML 1.0 Recommendation" described the initial goals for XML. These design goals, as listed on the W3C site, are:

- XML shall be straightforwardly usable over the Internet.
- XML shall support a wide variety of applications.
- XML shall be compatible with SGML.
- It shall be easy to write programs that process XML documents.
- The number of optional features in XML is to be kept to the absolute minimum, ideally zero.
- XML documents should be human-legible and reasonably clear.
- The XML design should be prepared quickly.

- ❑ The design of XML shall be formal and concise.
- ❑ XML documents shall be easy to create.
- ❑ Terseness in XML markup is of minimal importance.

XML was designed to fit where HTML was falling short (partly in extensibility and reusability) and where SGML had been more difficult than the average user could manage.

Understanding XML Basics

The basic unit of markup is an element. Elements can contain text, graphics paths, table data, and even other elements.

When you create an information model, you define an information hierarchy. The hierarchy describes the way in which elements can be used and how they can fit together. Your content is contained within this hierarchy and controlled somewhat by the elements used. As elements are nested hierarchically inside other elements, they take on a relationship to the elements that contain them. This relationship becomes useful in retrieving and sorting the data. Rules regarding the hierarchical nesting of elements include the following:

- ❑ The main element that is hierarchically around all other elements is called the root element.
- ❑ An element that has other elements hierarchically between its beginning and ending tags is the parent element to all elements immediately inside it, which are its child elements. (Elements a further level in would not be its child elements but its descendants.)
- ❑ An element can be a parent of other elements while being a child of one element. (Any parent of its parent, and so forth, would be considered its ancestors.)
- ❑ Child elements within the same parent are sibling elements. (There are no aunt, uncle, or cousin designations for elements.)

In an XML instance (document) representing a chapter of a book, for example, you might design a hierarchy in which a `<chapter_title>` child element always begins a `<chapter>`, where `<chapter>` is the root element. Structurally set as a sibling to the `<chapter_title>` element you might have an `<intro>` element followed by a `<numbered_list>` element. These siblings would be positioned beneath the `<chapter_title>` element, ensuring that the title information always appears first. The following XML markup example illustrates the structured hierarchy just described:

```
<chapter>
<chapter_title>Changing the Battery</chapter_title>
<intro>In this chapter, we review the procedures for changing a battery.</intro>
<numbered_list>
<item>Open the rear panel of the device.</item>
<item>Using a screwdriver, pry the old batteries out.</item>
<item>Insert the new batteries, using a hammer as needed to fit.</item>
</numbered_list>
.
.
[the rest of the document would be here]
.
</chapter>
```

In this example, <chapter> is parent to <chapter_title>, <intro>, and <numbered_list>. All the child elements of <chapter> are siblings. The <chapter> element is not a parent to any of the <item> elements; their parent is <numbered_list>. Position-wise it is more their "grandparent," but that isn't an official designation in XML, so you would refer to the <item> elements as "descendants" of the <chapter> element.

Because there is a hierarchy — relationship — between the chapter and its title, and between the intro and the chapter, those pieces of content could be retrieved and managed as a whole later on by using tools that can manipulate the markup. Then if a search tool found the chapter title and provided it in the search results, the person accessing the results could easily see that it was part of a chapter. You could even have an interface that showed the introductory paragraph, from the <intro> element, in search results before searchers retrieved the entire chapter.

Some elements appear as empty elements rather than as beginning and ending tags. These elements have no text content, but serve some purpose within the markup. For example,

```
<img src="imemine.jpg" />
```

is considered an empty element because the ending slash is inside the tag rather than in a separate end tag. An image is a fairly common example of an empty tag because there is generally a path to the image without any text content, only the tag and attribute data (as in this case).

Attributes are examined more later, but briefly an attribute is extra information about the element and is placed inside the beginning tag or in the empty tag. Attributes have a name, followed by a value in quotation marks. You will see more examples in the XML markup to come.

Exploring the Winery Markup Example

Take a closer look at the winery markup shown earlier:

```
<wine id="Muscadine_750ml">
<name>Muscadine</name>
<varietal>Scupp</varietal>
<vintage>1998</vintage>
<winery>Melvin</winery>
<bottlesize>750ml</bottlesize>
<description>Our complex purple Muscadine grape lends an assertive aroma and ↩
flavor, with a semi-sweet velvety smooth finish.</description>
</wine>
```

The markup is the angle-bracketed items (tags) placed around the content. The first line is the beginning tag of the root element <wine>:

```
<wine id="Muscadine_750ml">
```

For now, just ignore the id="Muscadine_750ml" metadata. The next line identifies the name of the wine:

```
<name>Muscadine</name>
```

The <name> beginning tag shows where the content begins, and the </name> end tag closes the element.

The varietal markup further identifies the wine and might be used later in sorting and selecting the wines for display:

```
<varietal>Scupp</varietal>
```

The vintage is critical data because wines are often selected based on their age:

```
<vintage>1998</vintage>
```

Assuming data from your winery and other wineries will be included in some master list by a partner or tourism agency, the winery name is included in the markup:

```
<winery>Melvin</winery>
```

This also enables sharing with directory publishers. Within the winery itself, this information most likely serves little purpose.

Listing the bottle size(s) available could be helpful in data sorting or business inventory management:

```
<bottlesize>750ml</bottlesize>
```

The <description> tag provides a marketing snippet describing the wine:

```
<description>Our complex purple Muscadine grape lends an assertive aroma and ↩
flavor, with a semi-sweet velvety smooth finish.</description>
```

This might help those who use this data to select the wine. It may be used directly in marketing pieces, or displayed in a web browser to reach those reading online. Potentially this could be combined with some keyword information to provide better searches and to better help searchers locate the best wine for them.

Finally, the end tag for the wine element that appeared as the very first line of the example closes out the document:

```
</wine>
```

That's enough XML markup information to get you going, and you'll better recognize the XML code to come. Now let's take a look at the winery example XML in more detail.

Determining an Information Model for the Winery XML

As you're learning, element markup in XML documents provides descriptive information about the content. It enables you to formalize your information models and create rules specific to the your content.

When you create an information model for your data, you identify all the pieces that compose the structure of your documents, and any hierarchical relationships between the pieces of content. In the preceding example, the markup identifies the content as parts of the <wine> element.

XML Elements

XML enables you to name and define each element you need and to assign additional pieces of metadata — attributes — to these elements. You can name these elements whatever you like. This highly descriptive markup then can be used to retrieve, reuse, and deliver content.

```
Here is an example of an XML document using element names defined for accounting
data.
<accounting>
<invoice>
<inv_number>8559</inv_number>
<inv_recipient>
<company_name>Mad Melvin Inc.</company_name>
<contact_name>Melvin Brock</contact_name>
<address1>100 Brockway Hill, Suite 1</address1>
<city>Kenmore</city>
<state>NY</state>
<zip>14223</zip>
</inv_recipient>
<amount>199.00</amount>
<due_date>2006/11/02</due_date>
</invoice>
</accounting>
```

Accounting data is quite different from the winery catalog data and may give you some idea of the endless possibilities for element naming and hierarchical arrangement. In this example, the `<invoice>` element is the only child of the `<accounting>` element, and is the parent of the `<inv_number>`, `<inv_recipient>`, `<amount>`, and `<due_date>` elements. The `<inv_recipient>` is parent to the elements `<company_name>`, `<contact_name>`, `<address1>`, `<city>`, `<state>`, and `<zip>`. All of the child elements of `<invoice>` and `<inv_recipient>` are descendants of the `<accounting>` element. These tags make it easy to call out all invoice amounts or due dates, or the recipient's address information.

When determining the best information model for your content, consider what information you have, how you want to use it, and what additional information must be added to your data set. Then create element markup and XML documents that meet your needs.

> *Do not strive to create a perfect structure because that may never be attained. You may not create a structure completely wrong or completely right; just aim for a structure that is logical and will do what you need it to do. Down the road, you may find it necessary to adjust the structure, and that's fine and planned for within the extensible design of XML.*

To ensure that your data is as complete as it needs to be, you need to analyze the situation not only before you begin but also periodically afterward. Whenever you find it necessary, you can modify your element set by removing elements, adding elements, renaming elements, or adjusting the hierarchy. Points for analysis are covered throughout this book.

XML Declaration

An XML document may begin with an XML declaration, which is a line that lets processors, tools, and users know that the file is XML and not some other markup language. Declarations are optional and need not be used. Here's what it looks like:

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

If a declaration is used, it must be at the top of the XML file, above the root element's beginning tag, like this:

```
<?xml version="1.0"?>
<accounting>
<invoice>
<inv_number>8559</inv_number>
<inv_recipient>
<company_name>Mad Melvin Inc.</company_name>
<contact_name>Melvin Brock</contact_name>
<address1>100 Brockway Hill, Suite 1</address1>
<city>Kenmore</city>
<state>NY</state>
<zip>14223</zip>
</inv_recipient>
<amount>199.00</amount>
<due_date>2006/11/02</due_date>
</invoice>
</accounting>
```

If you are producing XML instances that are small chunks of reusable data, you probably won't use declarations because they might end up elsewhere in the file as chunks are combined, and an XML declaration anywhere but first in the file will throw an error. If you have XML that will stand alone, and that might be handled by automated processes that verify if the files are XML, then you may want to include the XML declaration to ensure proper recognition of the files as XML documents.

Attributes and Information Modeling

XML elements can contain more than just their name inside their angle brackets. Elements can have *attributes*, which are additional bits of information that go with the element. The attributes appear inside the element's beginning tag.

For instance, the `<chapter>` element from an example earlier in the chapter can have an attribute of `author`. This means that the author's name, although not part of the document content, can be retained within the XML markup. The author's university affiliation could also be included. Attributes in the structure can be either required or optional. While there might always be an author named in the markup, some authors may not be affiliated with a university. In those cases, the name attribute would have a value (a name), but no university attribute would be added. These attributes will then enable you to locate information by searching the data for specific authors or universities. Here's an XML markup using the `author` attribute:

```
<chapter author="William Penn">
```

The end tag is still `</chapter>`. No attribute data is included in it.

Elements with multiple attributes have spaces between the attributes. Here's a `<chapter>` element with both `author` and `university` attributes:

```
<chapter author="William Penn" university="Duquesne">
```

Elements and their corresponding attributes are markup that enable you to create content with the structure you need. You can design elements and attributes the way you need them to be, creating dynamic content for delivery to multiple formats via various media, based on the preferences of your users.

Create an XML Document

Open Notepad or another text editor, and enter the following data:

```
<?xml version="1.0"?>
<wine id="Muscadine_750ml" type="dessert">
<name>Muscadine</name>
<varietal>Scupp</varietal>
<vintage>1998</vintage>
<bottlesize>750ml</bottlesize>
<description>Our complex purple <keyword>Muscadine</keyword> grape lends an ⮌
<keyword>assertive</keyword> aroma and flavor, with a semi-sweet velvety ⮌
<keyword>smooth</keyword> finish.</description>
</wine>
```

Remember, the book's pages aren't always wide enough for an entire code line. The ⮌ symbol indicates that you should not press Enter yet, but continue typing the following line(s). Press Enter at the end of the first line thereafter that does not have the ⮌ symbol.

For now, follow the example precisely. Chapter 3 explores the rules you must follow to create an XML document, after which you will know the basic syntax and can create your own XML documents from scratch. Following the rules results in your document being well-formed XML so that a parser — or browser — can use it.

Save your file as `muscadine.xml`. Your document will look much like the one shown in Figure 1-1.

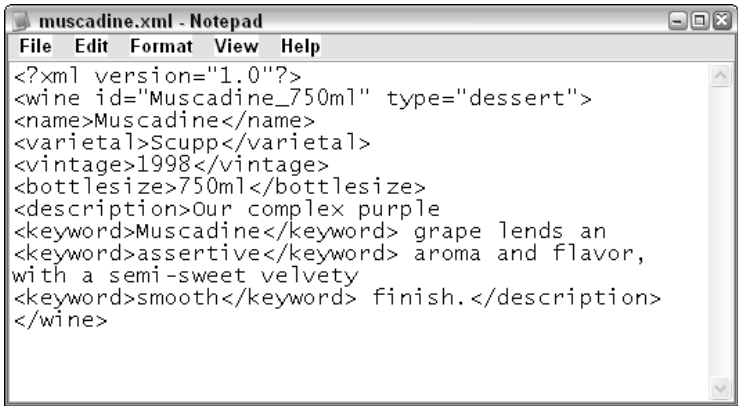

Figure 1-1

If your text editor adds `.txt` extension instead of an `.xml` extension, rename the file with an `.xml` extension before proceeding.

Now open your file in an XML-savvy web browser such as Microsoft Internet Explorer. You should see your elements in a tree view, with the hierarchical nesting clearly shown in each level. Your document should look similar to the one shown in Figure 1-2.

This is the first step in ensuring that your XML document is usable. If you cannot view it in a browser, it may have errors in the markup.

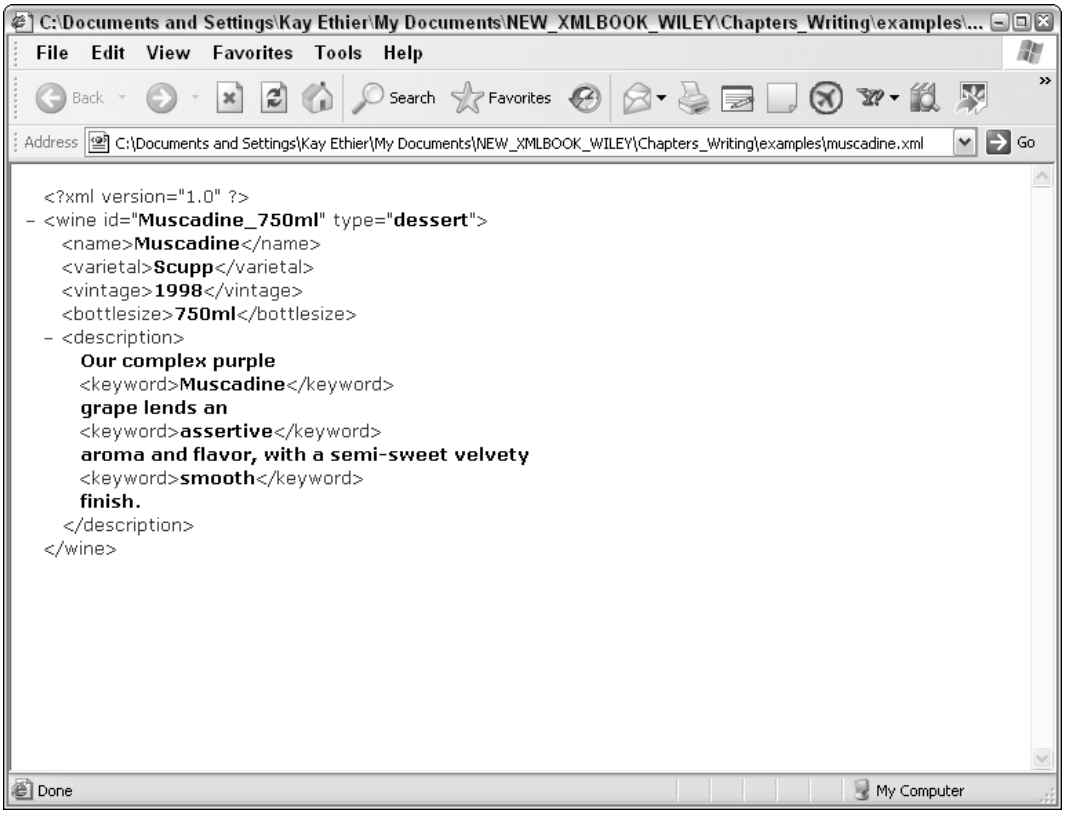

Figure 1-2

On your screen, you can see small minus signs next to the wine and description markup's beginning tags. All elements that have descendants appear with these minus signs next to them. You can click on these minus signs to hide the descendants. Once the descendents are hidden, the minus sign appears as a plus sign, which you can click to show the descendants and return the sign to a minus.

You may want your XML to be displayed a certain way. Most certainly, you will want it to look better than the tree view — unless it is being used by processes rather than being displayed.

XML can be formatted so that it looks good in a browser, using fonts, colors, and images instead of just showing in a tree view like the Internet Explorer view of Figure 1-2. You can make XML look good by using style sheets, such as the Cascading Style Sheets (CSS) used frequently for HTML. There is also an XML formatting and transformation language — XSLT (the Extensible Stylesheet Language Transformation — which you can use to produce XML or turn XML into another type of text-based document (you'll learn more about this later in the book).

Problems That XML Addresses

To better understand how XML can help your business projects, take a look at some of the ways it's used:

- ❑ Reusing content to multiple outputs or devices
- ❑ For text in multiple languages, as a way of tracking or managing translation among languages
- ❑ Enforcing structured authoring processes
- ❑ Enforcing data consistency standards
- ❑ Sharing nonproprietary data

Reusing Content (Multiple Outputs, Multiple Media)

If one of your goals is structured authoring, with or without added requirements for reusing your content or revision management, then XML authoring is a logical choice. XML authoring tools can help authors ensure that content rules are followed. XML documents can interact with databases, or even act as a database or repository for content chunks, so that structured content can be sorted, processed, extracted, and even automatically linked in for reuse in separate documents.

Content management software vendors are leading companies into XML by providing systems that enable you to accomplish what you want — and need — to do with your content. Content in the XML format is more easily managed than content in a basic text format or a word processing format because the XML is not bogged down with proprietary coding or formatting. Additionally, the XML information is identified by the elements used around the content. Content management systems enable those who want to create structured content to interface with more than one XML authoring tool, simplifying corporate purchases by avoiding protracted arguments over the "one tool" that must be used. With interfaces capable of interacting with multiple authoring tools, authors can create and share content without conversion, transformation, or other magic.

For example, a structured document can be saved to XML by the authoring tool, providing as a finished product an XML file with a complex hierarchy of data. Using a scripting language or stylesheet, chunks of the XML content can be pulled from one document and used to produce other documents. Additionally, data can be sorted and retrieved to create custom web pages, database content, or handheld device files. When changes are made to each linked content chunk, all of the documents that contain that chunk can be automatically updated.

Once your information is in XML and accessible, stylesheets and XSLT can be used to push your information to the web, PDF, cellular phones, iPod (PodCasts), and to some handheld devices. Publishing is not just for print anymore.

Managing Translated Content

Companies that publish in multiple languages often work with translation agencies. In many cases, word processor files are provided to the translation house, which translates each word into the target language(s) and formats the files for distribution. Service costs include translation and formatting (word processing) services. As an alternative, XML that can be translated without the need for additional formatting, which can amount to significant savings.

Because XML documents can include metadata — descriptive information about the document and the individual chunks that compose it — saving revision and language information within attributes or elements becomes straightforward. When sending an XML document for translation, metadata can be added by the client or the translation agency to indicate which pieces of the document require translation. In long documents, considerable savings can be realized by translating only new information in the document, rather than translating entire documents over and over.

Authoring with Enforced Structure and Automated Formatting

Authors working on the same document or document set can introduce inconsistencies due to different working styles. Even the same author sometimes makes different decisions about formatting. Additionally, multiple authors can choose to organize their content in slightly different fashions. Implementing a structured authoring environment based on XML provides an extra level of control — and hands-off formatting — that style guides and authoring "suggestions" cannot come close to duplicating.

Structured authoring guides authors to follow the content model of the structure being used. This type of enforcement eliminates the need for authors to make decisions about layout, format, and the order of things. Structured authoring enables you to formalize and enforce authoring, branding, and style guidelines, ensuring that the documents created are structurally correct and properly formatted.

In short, authors can write content with guidance and without concern for formatting.

Enforcing Consistency of Data

XML documents are consistently organized. Elements are named, defined, and tagged, and as a result, can be processed by other software applications that can read XML content. XML documents make content more easily retrievable and reusable.

XML documents can contain metadata designed to identify when an element was created, and by whom, thus improving your ability to ensure that you are providing the most up-to-date, accurate information available. When you spot an inaccuracy, you can fix it.

If, for example, an XML document outlines the menu items to be displayed in a software tool, the developers can consistently update the listing without fear of missing a location within the code. Similarly, strings of data used in error messages and other parts of the software interface can be controlled from a central point or be found easily by tracking within certain element structures.

Changes you later make to the source will be reflected immediately and automatically in all other areas of the product or product line that utilize that piece of information. In a perfect process, this information would also find its way to marketing.

The winery project includes many details for the wines listed. To ensure that this data is consistent and complete throughout, a data structure is created that checks the XML. Missing bits of data can be flagged in authoring or editing tools that understand or display XML. Data that is out of place or added can also be flagged, making error checking and correcting easier. This type of identification enables you to easily fix the data and begin using it again.

> *There are many tools on the market that allow the authoring or editing of XML. Tools used by the authors are mentioned briefly in the remaining chapters.*

Sharing Information

To facilitate the exchange of information, some industries have adopted common structures that formalize the structure of their information. By sharing common structures, industries' players are able to use a common vocabulary, which makes it easier, faster, and less expensive to exchange information with partners or clients.

Several industries were early adopters of structured authoring and shared structure; they have been reaping benefits for years already. Because these industry organizations shared data, and have invested heavily in structured content processes, they can insist that partners, affiliates, and customers integrate with their systems.

Airlines, for example, must integrate content provided by airplane parts manufacturers with their own documentation. While the parts manufacturers document the pieces of the airplane, the airlines create new information for pilot-training guides, user manuals, and the like, yet can integrate information about maintenance of parts easily by following a common structure. Airline parts manufacturers using a structured authoring approach can share their XML (or SGML) with their airline partners to ensure that both parties are using the same element names, metadata, and revision data.

Following is a sample of the type of information your winery could have about a partner company. This XML has more detail, including keywords marked up with a `<keyword>` element, than the simple examples shown earlier in this chapter.

```
<catalog>
<section subject="Wines">
<winery id="Melvin">
   <name>Mad Melvin's Muscadine Wines</name>
   <region>New York</region>
   <country>United States</country>
      <wine id="Nags_Head_Carlos_WHT_750ml">
         <name>Nags Head Carlos WHT</name>
         <varietal>Scupp</varietal>
         <vintage>1994</vintage>
         <bottlesize>750ml</bottlesize>
         <description>Aged in Carolina Willow Oak, this velvety ⮐
         <keyword>red</keyword> wine is highly complex, with a flavor of red ⮐
         <keyword>cherries</keyword>, <keyword>apricots</keyword> ⮐
          and <keyword>grapefruit</keyword>.</description>
       </wine>
       <wine id="Muscadine_750ml" type="dessert">
          <name>Muscadine</name>
          <varietal>Scupp</varietal>
          <vintage>1998</vintage>
          <bottlesize>750ml</bottlesize>
          <description>Our complex purple <keyword>Muscadine</keyword> grape⮐
           lends an <keyword>assertive</keyword> aroma and flavor, with⮐
           a semi-sweet velvety <keyword>smooth</keyword> finish.</description>
       </wine>
</winery>
</section>
</catalog>
```

The preceding markup is certainly helpful, although perhaps not as detailed as is needed for enterprise-wide use. However, because the structure provides basic information about each wine and includes the winery designation, it may be possible to access a partner's catalog, share your own winery catalog, and even combine the two catalogs. Assume that the winery data also must be shared with the tourism organizations that spread the word about wineries in the Finger Lakes region.

For this project, assume that the preceding markup is standard for the data your winery has made available.

To share this information properly, your company and any partner companies not only need the same XML document, but they would also ideally have the same structure or similar structures to allow sharing of information with little loss of dissimilar data or unused extra data.

To design a solution for combining the catalogs, you first review what you have and what your partner has, and then determine a plan for merging your data.

Here's a selection of the XML content of your partner's XML catalog of wine:

```
<?xml version="1.0" encoding="UTF-8"?>
<catalog>
   <section subject="Wines">
<distributor id="" name=""/>
      <distributor id="Aeg" name="Aegean Imports, INC."/>
      <distributor id="Cla" name="Classic"/>
      <distributor id="Cou" name="Country Vintner"/>
      <distributor id="Emi" name="Eminent Domains"/>
      <distributor id="Emp" name="Empire"/>
      <distributor id="Fran" name="Franklin Selection"/>
      <region name="Western Cape"/>
<winery id="">
         <region>Burgundy</region>
         <country>France </country>
      </winery>
      <winery id="ALong">
         <region>Chablis</region>
         <country>France</country>
      </winery>
    <wine id="Cabernet_Sauvignon_1992_750ml">
         <name>Cabernet Sauvignon</name>
         <varietal>Cabernet Sauvignon</varietal>
         <vintage>1992</vintage>
         <winery>Caskone</winery>
         <distributor>Aeg</distributor>
         <bottlesize>750ml</bottlesize>
      </wine>
      <wine id="Chardonnay_1994_750ml">
         <name>Chardonnay</name>
         <varietal>Chardonnay</varietal>
         <vintage>1994</vintage>
         <winery>Caskone</winery>
         <distributor>Aeg</distributor>
         <bottlesize>750ml</bottlesize>
      </wine>
      <wine id="Sauvignon_Blanc_1994_750ml">
         <name>Sauvignon Blanc</name>
         <varietal>Sauvignon Blanc</varietal>
         <vintage>1994</vintage>
         <winery>Caskone</winery>
         <distributor>Aeg</distributor>
         <bottlesize>750ml</bottlesize>
      </wine>
      <wine id="Zinfandel_15L">
         <name>Zinfandel</name>
         <winery>CorbetCanyon</winery>
```

```
            <distributor>Emp</distributor>
            <bottlesize>1.5L</bottlesize>
         </wine>
         <wine id="The_Cabernet_1996_750ml">
            <name>The Cabernet</name>
            <varietal>Cabernet Sauvignon</varietal>
            <vintage>1996</vintage>
            <winery>Cosentino</winery>
            <distributor>Emp</distributor>
            <bottlesize>750ml</bottlesize>
         </wine>
      </section>
   </catalog>
```

Take a closer look at the data available to you in this XML. The first line is, of course, the XML declaration:

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

The second line begins the company's catalog:

```
<catalog>
```

The end tag for the catalog element comes at the end of all the listings.

Although not particularly descriptive, `<section>` elements break the catalog information down by product, so that the wine-related data appears here and other nonwine products might be included at another point in the XML:

```
      <section subject="Wines">
```

Distributor data may help you reach distributors, and you may be pleasantly surprised to see that your partner company shared this data in its XML catalog:

```
         <distributor id="" name=""/>
         <distributor id="Aeg" name="Aegean Imports, INC."/>
         <distributor id="Cla" name="Classic"/>
         <distributor id="Cou" name="Country Vintner"/>
         <distributor id="Emi" name="Eminent Domains"/>
         <distributor id="Emp" name="Empire"/>
         <distributor id="Fran" name="Franklin Selection"/>
```

The `<region>` element is a sibling to the distributor data, and names the region; structurally, this is not well tied to the distributors or other data to come:

```
         <region name="Western Cape"/>
```

Multiple wineries are listed within the source, each with child element data designating the region and country. These `<winery>` elements are also siblings to the `<distributor>` elements.

```
         <winery id="">
            <region>Burgundy</region>
            <country>France </country>
         </winery>
```

```
<winery id="ALong">
   <region>Chablis</region>
   <country>France</country>
</winery>
```

After the winery data are the details of each wine, which include relational data back to the winery and distributor data:

```
<wine id="Cabernet_Sauvignon_1992_750ml">
   <name>Cabernet Sauvignon</name>
   <varietal>Cabernet Sauvignon</varietal>
   <vintage>1992</vintage>
   <winery>Caskone</winery>
   <distributor>Aeg</distributor>
   <bottlesize>750ml</bottlesize>
</wine>
<wine id="Chardonnay_1994_750ml">
   <name>Chardonnay</name>
   <varietal>Chardonnay</varietal>
   <vintage>1994</vintage>
   <winery>Caskone</winery>
   <distributor>Aeg</distributor>
   <bottlesize>750ml</bottlesize>
</wine>
<wine id="Sauvignon_Blanc_1994_750ml">
   <name>Sauvignon Blanc</name>
   <varietal>Sauvignon Blanc</varietal>
   <vintage>1994</vintage>
   <winery>Caskone</winery>
   <distributor>Aeg</distributor>
   <bottlesize>750ml</bottlesize>
</wine>
<wine id="Zinfandel_15L">
   <name>Zinfandel</name>
   <winery>CorbetCanyon</winery>
   <distributor>Emp</distributor>
   <bottlesize>1.5L</bottlesize>
</wine>
<wine id="The_Cabernet_1996_750ml">
   <name>The Cabernet</name>
   <varietal>Cabernet Sauvignon</varietal>
   <vintage>1996</vintage>
   <winery>Cosentino</winery>
   <distributor>Emp</distributor>
   <bottlesize>750ml</bottlesize>
</wine>
```

Some of the wines have more details included in the data than other wines have.

For the most part, your data structure matches that of your partner's catalog. Aside from including notations on distributors, the other markup uses the same element and attribute names as your snippets. It will be fairly easy to consolidate the like data so that your winery and this partner can share information.

At the end of the XML document is the closing information for the section (which included all wine product data) and the entire catalog:

```
    </section>
</catalog>
```

There is only one section in this file, so at this point the company has only structured its wine product data and not any peripheral offerings (apparel, foods, tours, and the like).

Solution

Starting with what you know of your winery content, expand the data beyond the basic information about the wines and add data necessary for enterprise use plus partner distribution. You change the XML content model in the next chapter.

Later, you will create a web interface to display the existing data and provide a form-based entry system for your wine data, which will enable you to fill in missing content and revise existing data easily.

Summary

XML is a format-free way to share information. Marking up content with XML enables you to identify what makes up your documents, and to identify each component potentially for reuse, sorting, or formatting. It is becoming more widespread for enterprise use as the tools and technologies expand.

You've been introduced to a number of concepts in this chapter, including:

- The basics of XML
- How to begin an information model
- How XML can resolve content and consistency problems
- Why XML is so helpful in sharing information

In Chapter 2, you learn the rules of XML markup and explore the concept of well-formed XML.