

# Methodology and Technology Services



www.ies.aust.  
Web com

[Home](#) | [Courses](#) | [Certification](#) | [Projects](#) | [Papers](#) | [EA Blog](#) | [Online Store](#)  
| [Contact Us](#) |

## XML - The Future of Metadata

[Printable PDF Version](#)

[Clive Finkelstein](#)

Extract from "[Building Corporate Portals with XML](#)"  
by Clive Finkelstein and Peter Aiken,  
McGraw-Hill (Sep 1999) [[ISBN: 0-07-913705-9](#)]

Copyright © 1999, The McGraw-Hill Companies, Inc. All rights reserved.

---

*This paper is based on an extract from the book: "[Building Corporate Portals with XML](#)", by Clive Finkelstein and Peter Aiken, published by McGraw-Hill in September 1999. The paper addresses one of the most significant developments of the Computer industry for the future. It shows how Metadata and Data Administration will shortly move into the mainstream and become one of the most important aspects of the WWW, and of systems development in general. The paper introduces the Extensible Markup Language (XML) – the successor to HTML for the Internet, for corporate Intranets and for Extranets. XML incorporates Metadata in any document, to define the content and structure of that document and any associated (or linked)*

[Home](#)  
[Courses](#)  
[Certification](#)  
[Projects](#)  
[Papers](#)  
[TEN Archive](#)  
[EA Blog](#)  
[Contact Us](#)  
[Search](#)  
[Links](#)  
[Online Store](#)

*resources. It has the potential to transform integration of structured data (such as in relational databases or legacy files) with unstructured data (such as in text documents, reports, email, graphics, images, audio and video files) for innovative application integration opportunities.*

---

Corporate Portals (also called Enterprise Portals - EPs) are based on Data Warehousing technologies, using *Metadata* and the *Extensible Markup Language* (XML) to integrate both structured and unstructured data throughout an enterprise. Metadata, XML and EPs will be vital elements of the 21<sup>st</sup> century enterprise.

*Structured data* exists in databases and data files that are used by current and older operational systems in an enterprise. We call these older systems *legacy systems*; we call the data they use *legacy data*. In most enterprises, structured data comprises only 10% of the data, information and knowledge resources of the business; the other 90% exists as *unstructured data* in textual documents, or as graphics and images, or in audio or video formats. These unstructured data sources are not easily accessible to Data Warehouses, but EPs use metadata and XML to integrate both structured and unstructured data seamlessly, for easy access throughout the enterprise.

## **1. What is Metadata and XML?**

IT staff in most enterprises have a common problem. How can they convince managers to plan, budget and apply resources for metadata management? What is metadata and why is it important? What technologies are involved? Internet and Intranet technologies are part of the answer and will get the immediate attention of management. XML is the other technology. The following analogy may help you outline to management the important role that metadata takes in an enterprise.

### **1.1 What is Metadata?**

Every country is now interconnected in a vast, global telephone network. We are now able to telephone anywhere in the world. We can phone a number, and the telephone assigned to that number would ring in Russia, or China, or in Outer Mongolia. But when it is answered, we may not understand the person at the

other end. They may speak a different language. So we can be connected, but what is said has no meaning. We cannot share information.

Today, we also use a computer and the World Wide Web. We enter a web site address into a browser on our desktop machine – a unique address in words that is analogous to a telephone number. We can then be connected immediately to a computer assigned to that address and attached to the Internet anywhere in the world. That computer sends a web page based on the address we have supplied, to be displayed in our browser. This is typically in English, but may be in another language. We are connected, but like the telephone analogy – if it is in another language, what is said has no meaning. We cannot share information.

Now consider the reason why it is difficult for some of the systems used in an organization to communicate with and share information with other systems. Technically, the programs in each system are able to be interconnected and so can communicate with other programs. But they use different terms to refer to the same data that needs to be shared. For example, an accounting system may use the term “customer” to refer to a person or organization that buys products or services. Another system may refer to the same person or organization as a “client”. Sales may use the term “prospect”. They all use different terminology – different language – to refer to the same data and information. But if they use the wrong language, again they cannot share information.

The problem is even worse. Consider terminology used in different parts of the business. Accountants use a “jargon” – a technical language – which is difficult for non-accountants to understand. So also the jargon used by engineers, or production people, or sales and marketing people, or managers is difficult for others to understand. They all speak a different “language”. What is said has no meaning. They cannot easily share common information. In fact in some enterprises it is a miracle that people manage to communicate meaning at all!

Each organization has its own internal language, its own jargon, which has evolved over time so similar people can communicate meaning. As we saw above, there can be more than one language or jargon used in an organization. Metadata identifies an organization’s own “language”. Where different terms refer to the same thing, a common term is agreed for all to use. Then

people can communicate more clearly. And systems and programs can intercommunicate with meaning. But without a clear definition and without common use of an organization's metadata, information cannot be shared effectively throughout the enterprise.

Previously each part of the business maintained its own version of "customer", or "client" or "prospect". They defined processes – and assigned staff – to add new customers, clients or prospects to their own files and databases. When common details about customers, clients or prospects changed, each redundant version of that data also had to be changed. It requires staff to make these changes. Yet these are all redundant processes making the same changes to redundant data versions. This is enormously expensive in time and people. It is also quite unnecessary.

The importance of metadata can now be seen. *Metadata defines the common language used within an enterprise so that all people, systems and programs can communicate precisely.* Confusion disappears. Common data is shared. And enormous cost savings are made. For it means that redundant processes (used to maintain redundant data versions up-to-date) are eliminated, as the redundant data versions are integrated into a common data version for all to share.

## 1.2 What is XML?

Much effort has earlier gone into the definition and implementation of *Electronic Data Interchange* (EDI) standards to address the problem of intercommunication between dissimilar systems and databases. EDI has now been widely used for business-to-business commerce for many years. It works well, but it is quite complex and very expensive. As a result, it is cost-justifiable generally only for large corporations.

Once an organization's metadata is defined and documented, all programs can use it to communicate. EDI was the mechanism that was used previously. But now this intercommunication has become much easier.

*Extensible Markup Language* (XML) is a new Internet technology that has been developed to address this problem. XML can be used to document the metadata used by one system so that it can be integrated with the metadata used by other systems. This

is analogous to language dictionaries that are used throughout the world, so that people from different countries can communicate. Legacy files and other databases can now be integrated more readily. Systems throughout the business can now coordinate their activities more effectively as a direct result of XML and management support for metadata.

XML now provides the capability that was previously only available to large organizations through the use of EDI. XML allows the metadata used by each program and database to be published as the language to be used for this intercommunication. But distinct from EDI, XML is simple to use and inexpensive to implement for both small and large organizations. Because of this simplicity, we like to think of XML as:

---

*"XML is EDI for the Rest of Us"*

---

XML will become a major part of the application development mainstream. It provides a bridge between structured and unstructured data, delivered via XML then converted to HTML for display in web browsers. Together with metadata, XML is a key component in the design, development and deployment of Enterprise Portals.

### **1.3 How Is Metadata Used with XML?**

Metadata is used to define the structure of an XML document or file. Metadata is published in a *Document Type Definition* (DTD) file for reference by other systems. A DTD file defines the structure of an XML file or document. It is analogous to the *Database Definition Language* (DDL) file that is used to define the structure of a database, but with a different syntax.

An example of an XML document identifying data retrieved from a PERSON database is illustrated in Figure 1. This includes metadata markup tags (surrounded by < ... >, such as <person\_name>) that provide various details about a person. From this, we can see that it is easy to find specific contact information in <contact\_details>, such as <email>, <phone>, <fax> and <mobile> (cell phone) numbers.

---

```

<PERSON person_id="p1100" sex="M">
  <person_name>
    <given_name>Clive</given_name>
    <surname>Finkelstein</surname>
  </person_name>
  <company>
    Information Engineering Services Pty Ltd
  </company>
  <country>Australia</country>
  <contact_details>
    <email>cfink@ies.aust.com</email>
    <phone>+61-8-9402-8300</phone>
    <phone>(08) 9309-6163</phone>
    <fax>+61-8-9402-8322</fax>
    <mobile>+61-411-472-375</mobile>
    <mobile>0411-472-375</mobile>
  </contact_details>
</PERSON>

```

**Figure 1:** An example of an XML document with metadata tags (surrounded by < ... >) identifying the meaning of following data

Although we have not shown it in Figure 1, the DTD can specify that certain tags must exist or are optional, and whether some tags can exist more than once – such as multiple <phone> and <mobile> tags above. XML is introduced in more detail later in this paper.

Metadata that is used by various industries, communities or bodies can be used with XML to define markup vocabularies. The World Wide Web Consortium (W3C) has developed a standard framework that can be used to define these vocabularies. This is called the *Resource Description Framework* (RDF). It is a model for metadata applications that support XML. RDF was initiated by the W3C to build standards for XML applications so that they can inter-operate and intercommunicate more easily, avoiding the communication problems that we discussed earlier.

With XML, many applications that were difficult to implement before – often due to metadata differences – now become possible. For example, an organization can define the unique metadata used by each supplier's legacy inventory systems. This enables the organization to place orders via the Internet directly with those suppliers' systems, for automatic fulfillment of product orders.

XML is enabling technology to integrate structured and unstructured data for next generation E-Commerce and EDI applications. Web sites will evolve to use XML, with far greater power and flexibility than offered by HTML. Netscape Communicator 5.0 and Microsoft Internet Explorer 5.0 browsers both support XML. Most productivity tools and office suites (such as Microsoft Office 2000) support XML. Business Intelligence and Knowledge Management tools will support XML. XML development tools are also being released so that XML applications can be developed more easily.

The acceptance of XML is progressing rapidly, as it offers a very simple – yet extremely powerful – way to intercommunicate between different databases and systems, both within and outside an organization. How well an organization accesses and uses its knowledge resources can determine its competitive advantage and future prosperity. Use and application of knowledge will become even more important in the competitive Armageddon of the Internet, in which we will all participate.

---

## **2. Transformations of the 1990s**

There have been three major transformations, or shifts, that have occurred in the Computer Industry throughout the 1990s. Their impact extends far beyond that industry. They are also transforming business and society. They are moving us rapidly from the Industrial Age to the Information Age.

### **2.1 The First Shift: The Internet**

The *First Shift* has already occurred: the impact that the World Wide Web is having on business today. With the introduction of web browsers in the early 90s, the Internet – already 20 years old at that time – moved into the mainstream as organizations rushed to establish their own web sites.

First generation web sites – using Hypertext Markup Language (HTML) – were used as billboards to the world. They provided static advertising and marketing information for the benefit of customers and suppliers. They implemented online information that was also available in print advertisements, or as documentation in book or manual formats. While effective with

those static media, when transferred to a web site they offered no benefit – only glitzy eye candy. These static web sites also suffered from another disadvantage. While they were easy to visit, they were also easy to leave with the click of a mouse – when potential customers could not find what they needed.

Second-generation web sites added interactivity and more content to provide further assistance. But alone, animated images or sounds and movie clips do not provide real benefit to visitors. They are still essentially "static" in their ability to bring real, bottom-line benefit to the business. They need to be integrated into the main purpose of the web site – as demonstration aids, sales aids or information aids for example. When they provide this purpose-focused capability, they move their web sites to the third generation.

Electronic Commerce sites that are extensively being established today are part of these third generation web sites. They have the potential to generate major revenue and profit for the business. But many of these electronic storefronts are like the Lemonade stands of our childhood – the first tentative ventures into a New World of business. More is needed before the full potential of Electronic Commerce can be realized.

## 2.2 The Second Shift: Java

The mid 90s saw the start of the *Second Shift*: the emergence of Java as a programming language able to be executed anywhere regardless of hardware platform or operating system. Java was first developed by a team lead by James Gosling at Sun Microsystems in 1991. It was planned as a portable language that could be executed from embedded devices such as TV set-top boxes. But its potential to become a major programming language that could transcend the hardware platform and operating system dependencies of other languages was also recognized. This saw the introduction by Sun in early 1995 of Java as a portable programming language. It was seen as the "Holy Grail of Computing": a hardware and operating-systems-independent language.

Java presented a potential threat to Microsoft, as it could offer an alternative operating environment to Windows and threaten its desktop monopoly. Microsoft therefore embraced Java, but it added extensions to use Windows-specific capabilities – so limiting the portability of the language. This was the subject of a suit brought by Sun against Microsoft in 1997, decided against

Microsoft in late 1998. The legal judgement required that Microsoft remove its Windows-specific Java extensions within 90 days of the ruling.

Java today is being adopted widely as a major object-oriented language across the industry. Java virtual machines are now available for all major operating system and hardware environments. Java compilers are also available for most operating systems: desktop, server and mainframe. The shift to Java is gathering steam, but it will be many years before its full promise of "write once, run anywhere" can be fully realized.

## **2.3 The Third Shift: Extensible Markup Language (XML)**

The Third Shift is the emergence of the Extensible Markup Language (XML) in the late 90s. This shift is just starting. It promises to be as significant as the first two. It has the ability to bring real, bottom-line benefits to business – in cost-reduction, in greater efficiency, in greater competition and in greater revenue.

XML is one of the most significant developments of the Computer industry since the World Wide Web and Java moved to their present positions of importance. For the next 2 - 5 years this will be one of the most important aspects of the Internet, and of systems development in general. It has the potential to move metadata and data administration also into the mainstream of systems development. XML will present major business opportunities, when used with the Internet, as a delivery channel for information from Data Warehouses and Enterprise Portals.

XML will be the successor to HTML for the Internet, Intranets, and for secure Extranets between customers, suppliers and business partners. XML incorporates metadata in any document, to define the content and structure of that document and any associated (or linked) resources. It has the potential to transform the integration of structured data (such as in legacy files or relational databases) with unstructured data (such as in text documents, reports, email, graphics and images, audio and video resources, and web pages). XML will be a significant technology for the deployment of Data Warehouses and Enterprise Portals.

XML uses the Extensible Style Language (XSL) and the Extensible Linking Language (XLL) to achieve this integration. XML, XSL and XLL allow the easy integration of dissimilar

systems for multiple worldwide customers and suppliers in any industry. It permits the ready integration of those systems, regardless of whether they are legacy systems and databases, Electronic Data Interchange (EDI) systems or Electronic Commerce. It represents the future direction of metadata and the important role that data administration will take in systems development in the years ahead.

There are steps that you can take now, to prepare today for the coming shift to XML.

## 2.4 Preparing for an XML World

XML assumes that your metadata has already been defined. This is necessary not only for the new systems that you want to develop, but also for the legacy systems and databases that you need to integrate with those new systems. XML will enable this integration to be carried out dynamically.

Data modeling and strategic modeling methods help you to define the metadata required by XML. These are Forward Engineering methods. They will also enable you to eliminate redundant data versions and redundant processes, to develop integrated databases for the Internet and Intranets. This is not just the responsibility of data administrators. It requires business knowledge also, gained by the active involvement of business experts.

A knowledge of the metadata types, metadata activities and metadata capture techniques using Reverse Engineering methods will also help you to extract the metadata from existing legacy systems and databases, or from relational or object databases. XML will enable you to combine reverse-engineered metadata with forward-engineered metadata, for the seamless integration of structured and unstructured data that characterizes truly effective Enterprise Portals.

Interest in XML, metadata and data administration will grow strongly. The XML specifications are now essentially complete [XML], while the XSL and XLL specifications were still evolving at the time of writing. These specifications are defined by the World Wide Web Consortium and are all available from their web site [W3C].

Some browser support for XML was first included in Microsoft

Internet Explorer 4.0. The Channel Definition Format (CDF) capability of Internet Explorer 4.0 was based on the use of XML. More complete support for XML is provided in Microsoft Internet Explorer 5.0 and Netscape Communicator 5.0. We will also see wide XML support added to DBMS products, to CASE tools, to Data Warehouse tools and also to Client / Server development tools. We will see a new generation of Knowledge Management tools evolve rapidly to take advantage of the structured/ unstructured data integration opportunities offered by XML.

Several books provide good treatment of XML. An initial introduction to XML (and also Cascading Style Sheets) is provided by "*XML: A Primer*" [St Laurent 1998]. XML used for web site development, with HTML, XSL and XLL, is addressed in "*XML: Extensible Markup Language*" [Harold 1998]. "*XML Complete*" [Holzner 1998] covers the use of XML with Java. These can be used as detailed references for XML. "*Web Farming for the Data Warehouse*" [Hackathorn 1998] uses the Internet, Intranets and XML for access to external data sources for warehouse deployment.

We will now examine XML concepts. In a short paper, of necessity this can only be an overview, and it ignores any treatment of XSL and XLL. They are all covered in greater detail in "*Building Corporate Portals with XML*" [Finkelstein 1999]. More detail is also available from the references above. We will start with the initial purpose of XML, which was to provide a more effective capability for defining document content than that offered by HTML.

## 2.5 Some Problems using HTML

Tim Berners-Lee at CERN, the originator of the Word Wide Web (WWW) in 1990, developed Hypertext Markup Language (HTML) as a subset of the Standard Generalized Markup Language (SGML). A standard for the semantic tagging of documents, SGML evolved out of work done by IBM in the 1970s. It is used in Defense and other industries that deal with large amounts of structured text. SGML is powerful, but it is also very complex and expensive.

HTML was defined as a subset of SGML – specifically intended as an open architecture language for the definition of WWW text files transmitted using Hypertext Transport Protocol (HTTP) across the Internet. HTML defines the layout of a web page to a web browser running as an open architecture client. Microsoft

Internet Explorer and Netscape Communicator share over 90% of the web browser market; both are now available free.

An HTML page contains text as the content of a web page, as well as tags that define headings, images, links, lists, tables and forms to display on that page. These HTML tags also contain attributes that define further details associated with a tag. An example of such attributes is the location of an image to be displayed on the page, its width, depth and border characteristics, and alternate text to be displayed while the image is being transmitted to the web browser.

Because of this focus on layout, HTML is recognized as having some significant problems:

1. **No effective way to identify content of page:** HTML tags describe the layout of the page. Web browsers use the tags for presentation purposes, but the actual text content has no specific meaning associated with it. To a browser, text is only a series of words to be presented on a web page for display purposes.
2. **Problems locating content with search engines:** Because of a lack of meaning associated with the text in a web page, there is no automatic way that search engines can determine meaning – except by indexing relevant words, or by relying on manual definition of keywords.
3. **Problems accessing databases:** Web pages are static. But when a web form provides access to online databases, that data needs to be displayed dynamically on the web page. Called “Dynamic HTML” (DHTML), this capability enables dynamic content from a database to be incorporated “on-the-fly” into an appropriate area on the web page.
4. **Complexity of dynamic programming:** DHTML requires complex programming to incorporate dynamic content into a web page. This may be written as CGI, Perl, ActiveX, JavaScript or Java logic, executed in the client, the web server, the database server, or all three.
5. **Problems interfacing with back-end systems:** This is a common problem that has been with us since the beginning of the Information Age. Systems written in one programming language for a specific hardware platform, operating system

and DBMS may not be able to be migrated to a different environment without significant change or a complete rewrite. Even though it is an open architecture, HTML also is affected by our inability to move these legacy systems to new environments.

Recognizing these limitations of HTML, the W3C SGML working group (now called the XML working group) was established in mid 1996. The purpose of this group was to define a way to provide the power of SGML, while also retaining the simplicity of HTML. The XML specifications were born out of this activity [XML].

XML retains much of the power and extensibility of SGML, while also being simple to use and inexpensive to implement. It allows tags to be defined for special purposes, with metadata definitions embedded internally in a web document – or stored separately as a Document Type Definition (DTD) script. A DTD is analogous to the Data Definition Language script (DDL) used to define a database, but it has a different syntax.

As we discussed earlier, data modeling and metadata are key enablers in the use and application of XML. The Internet and Intranets allow us to communicate easily with other computers. Java allows us to write program logic once, to be executed in many different environments. But these technologies are useless if we cannot easily communicate with and use existing legacy systems and databases.

We discussed earlier that we can now make a phone call, instantly, anywhere in the world. The telephone networks of every country are interconnected. When we dial a phone number, a telephone assigned to that number will ring in Russia, or China, or Outer Mongolia, or elsewhere. It will be answered, but we may not understand the language used by the person at the other end.

So it is also with legacy systems. We need more than the simple communication between computers afforded by the Internet. True, we could rewrite the computer programs at each end in Java, C, C++, or some other common language. But that alone would not enable effective and automatic communication between those programs. Each program must know the metadata used by the other program and its databases so that they can communicate with each other.

Considerable work has been carried out to address this problem. Much effort has gone into definition and implementation of Electronic Data Interchange (EDI) standards. EDI has now been widely used for business-to-business commerce for many years. It works well, but it is complex and expensive. As a result, it is cost-justifiable generally only for larger corporations.

XML now also provides this capability. It allows the metadata used by each program and database to be published as the language to be used for this intercommunication. But distinct from EDI, XML is simple to use and inexpensive to implement. XML will become a major part of the application development mainstream. It provides a bridge between structured databases and unstructured text, delivered via XML then converted to HTML during a transition period for display in web browsers. Web sites will evolve over time to use XML, XSL and XLL natively to provide the capability and functionality presently offered by HTML, but with greater power and flexibility. XML components are listed in Table 1.

---

**Table 1:** *Components of XML*

Acronym	Name	Description
XML	Extensible Markup Language	Defines document content using metadata tags and namespaces
DTD	Document Type Definition	Defines XML document structure (analogous to DDL schema)
XSL	Extensible Style Language	XSL or Cascading Style Sheets (CSS) separate layout from data
XLL	Extensible Linking Language	XLL implements multi-directional links (single or multiple)
DOM	Document Object Model	Implements a standard API for processing XML in any language
RDF	Resource Description Framework	W3 Interoperability Project for data content interchange

The rest of this paper provides an introduction to XML and

DTDs, with only brief reference to XSL, XLL, DOM and RDF. Further information in each of these areas can be obtained from the book and web site references provided at the end of the paper.

---

### 3. A Simple XML Example

We will start our introduction to XML with a customer example in Figure 2. This illustrates some basic XML concepts. It shows customer data (in italics), such as entered from an online web form or accessed from a customer database. It shows the inclusion of metadata "tags" (surrounded by < and >) – such as <customer\_name>.

The tag: <customer\_name> is a start tag; the text following it is the actual content of the customer name: *XYZ Corporation*. It is terminated by an end tag: the same tag-name, but now preceded by "/" – such as </customer\_name>. Other fields define <customer\_address>, <street>, <city>, <state> and <postcode>. Each of these tags is also terminated by an end tag, such as </street>, </city>, </state> and </postcode>. The example concludes with </customer\_address> and </CUSTOMER> end tags.

```
<CUSTOMER>
  <customer_name>XYZ Corporation</customer_name>
  <customer_address>
    <street>123 First Street</street>
    <city>Any Town</city>
    <state>WA</state>
    <postcode>12345</postcode>
  </customer_address>
</CUSTOMER>
```

**Figure 2:** A Simple XML Example

From this simple example of XML metadata, we can see how the meaning of the text between start and end tags is clearly defined. We can also see that search engines can use these definitions for more accuracy in identifying information to satisfy a specific query.

Even more effective applications become possible For example,

an organization can define the unique metadata used by its suppliers' legacy inventory systems. This will enable that organization to place orders via the Internet directly with those suppliers' systems, for automatic fulfillment of product orders. XML is enabling technology to integrate unstructured text and structured databases for next generation E-Commerce and EDI applications.

The following pages now examine the XML syntax in more detail.

### 3.1 XML Naming Conventions

An XML document must be "well formed". To be well formed, a document must obey the following rules:

§ A tag name must start with a letter or underscore, with no spaces. Thus "*person\_id*" is correct, but not "*person id*" or "*1st name*".

§ XML names are case sensitive. For example, "*PERSON*", "*Person*" and "*person*" are all different names.

§ Each tag must have surrounding < and > indicators, as in the start tag *<tag\_name>*.

§ Each start tag must also have an end tag, as in *</tag\_name>*.

§ If a tag is empty, it must still have an end tag or empty tag such as *<CUSTOMER></CUSTOMER>* or *<country/>* (i.e. Empty).

§ Attribute values are preceded by an = sign and are surrounded by double or single quotes, such as *version="1.0" standalone="YES"*.

§ The characters <, >, &, " or ' cannot be used in XML except when replaced by their "escaped" versions. Thus the character string *&lt;* represents "<" at all times until it is to be displayed. Similarly *&gt;* is ">", *&amp;* is "&", *&quote;* is """ and *&apos;* is "'". These character sequences are called "predefined entity references".

A well-formed XML document example follows in Figure 3,

similar to the earlier XML example in Figure 1.

```

    <PERSON person_id="p1100" sex="M">           (Attributes
in Element)
        <person_name>                           (Children of
            <given_name>Clive</given_name>
"person_name"
            <surname>Finkelstein</surname>     Element)
        </person_name>
        <email>cfink@ies.aust.com</email>
        <company>
            Information Engineering Services Pty Ltd
        </company>
        <country>Australia</country>
        <phone>+61-8-9402-8300</phone>
        <fax>+61-8-9402-8322</fax>
    </PERSON>

```

**Figure 3:** Example of a Well-Formed XML Document

Notice that double quote characters in Figure 3 surround the attribute values of *PERSON*, declared on the first line with the values: `person_id="p1100" sex="M"`.

### 3.2 The XML Document Prolog

Every XML document starts with an XML declaration as part of its prolog. This declaration must be the first statement on the first line of the document. It is defined as a processing instruction (surrounded by `<? ... ?>` tags) such as:

```

    <?xml version="1.0" standalone="yes" encoding="Unicode"?
>

```

The `<?xml` specifies that the document uses XML syntax. An XML parser or application can analyze the content of the document prior to it being processed. The tag "XML", "xml" or any upper and lower case combination of this sequence of letters is reserved and cannot be used in any tag name.

The version number is specified for compatibility with future XML versions. The standalone specification indicates whether a Document Type Definition is included in-line ("standalone=yes") or out-of-line in an external file ("standalone=no"). We will discuss this shortly in relation to *DOCTYPE Declarations*.

The “encoding” statement specifies the language-encoding format used by the XML document. XML has been defined so it can be used with any language, such as English and European languages, as well as double byte Asian languages – Japanese, Chinese or Korean.

### 3.3 DOCTYPE Declarations

A Document Type declaration (“DOCTYPE”) immediately follows the `<?XML ... ?>` statement. Every XML document contains a root name, which includes all other XML tag names. The DOCTYPE statement identifies the specific root name used by the document. It also identifies the location of the Document Type Definition (DTD) file that is to be used with the document.

A DOCTYPE declaration has the following formats, with examples:

```
<!DOCTYPE root_element_name [ ... ]>
```

OR

```
<!DOCTYPE root_element_name SYSTEM “DTD_URL”>
```

1. `<!DOCTYPE CUSTOMER [ ... ]>`
2. `<!DOCTYPE CUSTOMER SYSTEM “customer.dtd”>`
3. `<!DOCTYPE supplier PUBLIC “http://www.ind-xml.com/supplier.dtd”>`

The first example specifies that the DOCTYPE is declared internally in the same document. We will see an example of this format shortly.

The second example declares that an external DTD is used as a private file (“SYSTEM”). It is the DTD file that is located at the relative Uniform Resource Locator (URL) “*customer.dtd*” within the same web site directory.

The third example specifies that the DTD is PUBLIC. It is the DTD file at the absolute URL “*http://www.ind-xml.com/supplier.dtd*”.

### 3.4 URL and URI

These DOCTYPE examples use relative or absolute URLs to identify the location of an external DTD file. But files and other resources can be moved to different URL locations. With HTML web pages, every link that refers to a moved resource must be updated to refer to its new URL. HTML links can be from web sites anywhere in the world. These can all refer to the same URL. Relocating a resource to a different URL can therefore require considerable maintenance work.

To overcome this problem, in time XML and XLL will enable resources to be located instead by a Uniform Resource Identifier (URI). Distinct from a URL, a URI can never change. XLL, with XLinks and XPointers, define a URI. The URI always points to that resource.

### 3.5 XML Comments

Comments can be used in an XML document to describe the purpose, intent and use of different statements. Comments can also document and separate logical sections of a document.

Comments in XML are defined similarly to HTML comments, surrounded by `<!-- ... -->` tags. For example:

```
<!-- This is a comment and is not processed -->
```

Comments can contain any data except the literal `-->` but may not be placed inside an XML tag. In the next two examples, the first comment is wrong; the second comment is correct:

```
<customer_name <!-- Defines customer name --> >
(incorrect)
```

```
<!-- Defines customer name --> <customer_name>
(correct)
```

However comments can be used to surround and hide tags, such as:

```
<!-- The following tag is used only for retail customers
<retail-code>2</retail-code>
and is ignored for wholesale customers -->
```

An XML parser or XML application cannot process the <retail-code> tag until the surrounding comment is removed, or until the tag is moved outside the comment. While it remains within the comment, for XML processing purposes the tag does not exist.

### 3.6 Processing Instructions

Processing instructions (PIs) declare applications that will be used to process part (or all) of an XML document. Like comments, they are not part of the XML document. An XML processor must pass a PI unchanged to the relevant XML application. A PI has the format:

```
<?PI_target-name PI_data?>
```

The *PI\_target\_name* identifies the application. *PI\_data* following the *PI\_target\_name* is optional; it is specified by and used by the PI application. We saw a PI example in *XML Document Prolog*. A document to be processed by an XML parser or processor was declared by the PI statement:

```
<?xml version="1.0" standalone="yes" encoding="Unicode"?  
>
```

XML applications should process only the targets they recognize. PI names that begin with "XML", in any combination of upper or lower case, are reserved for use in XML standards. PIs are used for document-specific application processing.

---

## 4. XML Elements, Attributes and Entities

XML defines metadata tags using elements, attributes and entities. In the following sections we will learn how XML uses these to declare metadata tags.

### 4.1 Declaring XML Elements

The tags that we have seen are all examples of XML "elements". An element is a named metadata tag that is declared in a DOCTYPE statement. As we have seen, a DOCTYPE can be defined externally in a DTD file, located using a relative or

absolute URL. Alternatively, a DOCTYPE can be defined internally. It is included in-line, immediately following the XML processing declaration. Figure 4 shows an internal declaration of the DOCTYPE statement that is used with the customer example in Figure 2:

```
<?XML version="1.0" standalone="YES"?>
<!DOCTYPE CUSTOMER
[
<!ELEMENT CUSTOMER ANY>
<!ELEMENT customer_name (#PCDATA)>
<!ELEMENT customer_address EMPTY>
<!ELEMENT street (#PCDATA)>
<!ELEMENT city (#PCDATA)>
<!ELEMENT state (#PCDATA)>
<!ELEMENT postcode (#PCDATA)>
]>
```

**Figure 4:** DOCTYPE Declaration for the Customer Example in Figure 2

The DOCTYPE statement in Figure 4 defines the XML root name as <CUSTOMER>. Square left and right brackets ([ ... ]) follow, surrounding all element declarations. The DOCTYPE root name <CUSTOMER> is declared as an ELEMENT, specified to be ANY (case-sensitive). This indicates that any element, as well as parsed character data (shown as italics in Figure 2) can appear in a <CUSTOMER> element.

Each element must be uniquely named within an XML document. As a DOCTYPE declaration can be defined using more than one DTD, the concept of namespaces has been included in XML. This allows an alias to be assigned to a DTD. The namespace alias can then be used to qualify named elements that would otherwise violate this rule, so ensuring uniqueness.

The declaration of an XML Element in Figure 4 has the format:

```
<!ELEMENT element-name content_type>
```

As with all XML tags, the *element-name* starts with a letter or underscore, can have no spaces and all names are case sensitive. Thus *Customer*, *customer* and *CUSTOMER* are different XML element-names. Because of this and to avoid confusion, it is recommended that an element-name be declared

using a case-sensitive name that always refers to that same element. Once declared, that *element-name* is used as the *tag-name*; the terms *element-name* and *tag-name* are therefore synonymous.

The *content\_type* of an Element can have values of ANY, EMPTY, (#PCDATA), or a (Child List) as discussed next.

We discussed an example of ANY in relation to the root name element <CUSTOMER> in Figure 4. This indicates that any element, as well as parsed character data, can appear within it.

By default, tags are *non-empty* and are followed by data (see italics in Figure 2). An element is declared EMPTY if it normally has no data. For example, the element <customer-address> in Figure 2 contains <street>, <city>, <state> and <postcode> elements within it. These are called child elements. As the parent element, the data for <customer-address> is provided by its child elements. The <customer-address> element is therefore declared in Figure 4 to be EMPTY.

Figure 4 declares the <customer\_name> element is (#PCDATA). This specifies that the element contains "Parsed Character Data". In Figure 2, we now know that *XYZ Corporation* is character data that is parsed by an XML parser, or processed by an XML application.

Similarly, the <street>, <city>, <state> and <postcode> elements in Figure 4 are also declared to be (#PCDATA). Note that <postcode> in Figure 2 contains the numeric characters: "12345", not the numeric value. For example, consider the following element declaration and corresponding tag:

```
<!ELEMENT customer_balance (#PCDATA)>
```

```
... ..
```

```
<customer_balance>$15,500.00</customer_balance>
```

Before the <customer\_balance> data of \$15,500.00 can be processed by an XML application, it must first be converted from the numeric characters "\$15,500.00" to the numeric currency value of \$15,500.00.

When we consider address data in an application, there are many variations. For example, in addition to the street number and name, some customers may have a floor or level number, and/or an apartment, suite or flat number. We could define each of these as separate elements within <CUSTOMER>. But this data could be considered as part of the normal content for the <street> element. Some customers may need two or more <street> elements. For others, <postcode> may not be available and so could be omitted.

To this point there is nothing in the declaration of CUSTOMER to control whether any or all of the declared elements must exist. We provide extra control by specifying a *content model*.

```
<?XML version="1.0" standalone="YES"?>
  <!DOCTYPE CUSTOMER
  [
    <!ELEMENT CUSTOMER ANY>
    <!ELEMENT customer_name (#PCDATA)>
    <!ELEMENT customer_address (street, city, state,
postcode)>
      <!ELEMENT street (#PCDATA)>
      <!ELEMENT city (#PCDATA)>
      <!ELEMENT state (#PCDATA)>
      <!ELEMENT postcode (#PCDATA)>
    ]>
```

**Figure 5: Constraints added to the CUSTOMER DOCTYPE Declaration**

Figure 5 indicates that <customer\_address> has a (*child list*). This child list is a content model, which specifies that <customer\_address> has child elements of (*street, city, state, postcode*). This comma-delimited format indicates that each customer address has only one <street>, <city>, <state> and <postcode> element. Each element must appear in the specified sequence.

When a child list is defined using commas, each element is mandatory and must exist in that sequence. Alternatively, if any elements validly may not exist, they can be separated by “|” to indicate optionality, such as:

```
<!ELEMENT customer_address (street | city | state | postcode)
>
```

If we also add a `<contacts>` element, with child elements of `<phone>`, `<fax>`, `<mobile>` (cell phone) and `<email>` elements we find even more variations. We therefore need to be able to specify the number of occurrences of child elements that are valid.

```
<?XML version="1.0" standalone="YES" ?>
  <!DOCTYPE CUSTOMER
  [
    <!ELEMENT CUSTOMER ANY>
    <!ELEMENT customer_name (#PCDATA)>
    <!ELEMENT customer_address (street+, city, state,
postcode?)>
      <!-- ? = zero or one; * = zero or more; + = one or more --
>
    <!ELEMENT street (#PCDATA)>
    <!ELEMENT city (#PCDATA)>
    <!ELEMENT state (#PCDATA)>
    <!ELEMENT postcode (#PCDATA)>
    <!ELEMENT contacts (phone+, fax*, mobile?, email?)>
      <!ELEMENT phone (#PCDATA)>
      <!ELEMENT fax (#PCDATA)>
      <!ELEMENT mobile (#PCDATA)>
      <!ELEMENT email (#PCDATA)>
    ]>
```

**Figure 6:** Further constraints added to the CUSTOMER DOCTYPE declaration for customer\_address and contacts.

Figure 6 adds validity constraints to child elements by including a suffix character attached to the child name. A suffix of “?” specifies that zero or one occurrence of the child element may exist within the parent element. A suffix of “\*” specifies that zero or more occurrences may exist, while a suffix of “+” specifies that at least one or more occurrences of the relevant child element must exist within the parent element. No suffix indicates that the element must exist only once.

Examining Figure 6 further, we see that `<customer_address>` must have at least one `<street>` element, but it can have more `street` occurrences (`street+`). There must be only one `<city>` and one `<state>` element (no suffix). But the `<postcode>` element is optional; there may be none, or one occurrence (`postcode?`). The comma delimiters specify that the elements must appear in the declared sequence.

We may want to show that a valid *customer\_address* can have several addresses within it. We can place these child elements all within brackets, with the relevant group suffix character following the right bracket. We also use surrounding brackets to group other elements.

```
<!ELEMENT customer_address
  (street+ | (city, state) | postcode)+>
```

The above fragment indicates by outer brackets with a suffix “+” that there must be at least one or more group of addresses. Within an address group, there must be one or more *street* elements (*street+*) OR a *city* element followed by a *state* element OR a *postcode* element). Of course, all elements can also exist in the above example.

In Figure 6 we also saw new element declarations of *<contacts>*: *<phone>*; *<fax>*; *<mobile>*; and *<email>*. We see that *<contacts>* has a content model with child elements of (*phone+*, *fax\**, *mobile?*, *email?*)>.

Based on the suffix attached to each of the *<contacts>* child names, Figure 6 specifies that there must be at least one or more *<phone>* occurrences (*phone+*) and zero or more *<fax>* occurrences (*fax\**). There can be zero or one *<mobile>* occurrence (*mobile?*), and also zero or one *<email>* occurrence (*email?*).

We can use a content model that includes PCDATA. For example, we can alternatively specify *<phone>* and *<fax>* by the fragment:

```
<!ELEMENT phone (#PCDATA | (country-code, area-code,
phone-number))*>
<!ELEMENT fax (#PCDATA| (country-code, area-code, phone-
number))*>
  <!ELEMENT country-code (#PCDATA)>
  <!ELEMENT area-code (#PCDATA)>
  <!ELEMENT phone-number (#PCDATA)>
```

There can be zero or more *<phone>* and *<fax>* – by the suffix “\*” after the outer brackets. These can contain parsed character data, or they may optionally have a child element group in the sequence of (*<country-code>*, *<area-code>* and *<phone-number>*). All content models that include PCDATA must have

this format: PCDATA must come first, vertical bars must separate all elements or element groups, and the entire outer group must be optional.

These constraints enable an XML Parser or XML application to confirm the validity of the document, by checking the number of child element occurrences within each parent element. They validate these occurrences against those specified by the child list constraints for the parent element in an internal DOCTYPE declaration, or a DOCTYPE in an external DTD.

## 4.2 Declaring XML Attributes

An element may contain one or more attributes to provide additional details about that element. Figure 3 earlier included an example of attributes for the PERSON element. This specified a PERSON occurrence, with a unique identification attribute called *person\_id* and another attribute called *sex*, repeated now in Figure 7.

```
<PERSON person_id="p1100" sex="M">
```

**Figure 7:** *The PERSON Element, with Attributes*

This example shows that attributes and their values are enclosed within the < and > characters of the start tag for an element, immediately following the element name.

Each attribute of an element is declared in DOCTYPE ATTLIST, using the format in Figure 8. The ATTLIST format specifies the *element\_name* and then defines an *attribute\_name* as a unique XML name within all of the element's attributes. It observes all of the rules detailed earlier in *XML Naming Conventions*.

```
<!ATTLIST element_name attribute_name type "default value">
```

Where type = (CDATA | ID | IDREF |  
IDREFS | ENTITY | ENTITIES | NMTOKEN |  
NMTOKENS | NOTATION)

**Figure 8:** *The ATTLIST Format*

The *type* specification in Figure 8 is defined from an enumerated

list of valid values: (CDATA | ID | IDREF | IDREFS | ENTITY | ENTITIES | NMTOKEN | NMTOKENS | NOTATION).

CDATA represents "Character Data", as a character data type that is non-markup text. This is somewhat analogous to a Data Definition Language (DDL) SQL data type of VARCHAR, as used by DBMS products.

Note that XML does not support the other DDL data types such as *numeric* or *decimal* (with a defined length and precision), or *money*, *currency* or *CHAR* (with a defined length), or *float*, *bit*, *Boolean* or other data types. XML is used and read as text. An XML application must convert and validate these other data types.

We will continue with the other *type* declarations for Figure 8. ID represents an identifying attribute such as a primary key, with a unique name within the element. There can only be one attribute in an element that is specified with a *type* of ID.

Where an element must have a compound primary key for uniqueness, a single unique primary key is defined. In data modeling this is called a "surrogate key". The compound primary keys are instead defined as foreign keys with a *type* of IDREF or IDREFS. These are discussed shortly.

Furthermore, the value of each ID attribute must be unique for all occurrences of the relevant element. This follows the uniqueness rule of primary keys: a primary key cannot have duplicates. The earlier PERSON example has a unique value of "p1100" for the attribute named *person\_id*, repeated as Figure 7.

An attribute can be defined as a foreign key, with a *type* of IDREF. Or several attributes can all be specified as foreign keys, each with a *type* of IDREFS. This offers more flexibility. It is used to specify many foreign keys. The referenced IDREF attribute name must also exist elsewhere, in an element where it is also declared as an ID or IDREF attribute. As we discussed earlier, IDREF or IDREFS can be used to specify compound primary keys, where a single primary (surrogate) key is specified with a *type* of ID.

In Figure 8 the *type* declarations ENTITY and ENTITIES define an attribute name, or attribute names, with associated substitution text. These declare entity references. The defined

entity name can be used as a shorthand notation, analogous to a macro; it is replaced by the substitution text wherever it is used. Entities can be used within the main body of the XML document, or in a DTD. We will cover *Entity Declarations* shortly.

Note that the use of ENTITY and ENTITIES by XML is different to the use of these terms in data modeling and normalization.

NMTOKEN and NMTOKENS *types* specify that the value of an attribute must be a valid XML name (NMTOKEN) or valid multiple XML names (NMTOKENS). A program can use an attribute of this *type* to manipulate XML data. For example, it can be used to associate a Java class with an element. A Java API can then be used to pass the data to a method for that class.

A NOTATION *type* typically is used to specify an application to process an unparsed value of an attribute. A NOTATION attribute is associated with a NOTATION declaration in a DTD. This declares the specific application program name to be invoked. We saw earlier that applications can be declared in a Processing Instruction (PI). This declares the *PI\_target\_name* as the application, with associated *PI\_data*.

We will now discuss the “*default value*” specification in Figure 8. This is used to define a list of valid values for an attribute, or it can declare an attribute as being *#REQUIRED*, *#IMPLIED* or *#FIXED*.

For example, attributes of PERSON in Figure 7 are specified by an ATTLIST declaration in Figure 9. We can see that *person\_id* is an ID attribute. Every PERSON occurrence must have a unique *person\_id* value. Further, this ID attribute is mandatory (*#REQUIRED*).

```
<!ELEMENT PERSON EMPTY>
  <!ATTLIST PERSON person_id ID #REQUIRED>
  <!ATTLIST PERSON sex (M | F) #IMPLIED>
  <!ATTLIST PERSON status (employee | trainee)
“employee”>
  <!ATTLIST PERSON company CDATA #FIXED
“XYZ”>
```

**Figure 9:** A List of Valid Attribute Values

The attribute *sex* in Figure 9 has valid values of “M” (Male) or

"F" (Female). Any other values are invalid. This attribute example is #IMPLIED. It is not mandatory for a value to be supplied. The sex attribute can be omitted if it is not known.

A default value can be provided if an attribute is able to be omitted. In the example, *status* can only have valid values of "employee" or "trainee". If not specified, *status* defaults to "employee".

Finally, an attribute can be declared as #FIXED. This allows a default value to be supplied for an attribute, which cannot be changed. Figure 9 shows that *company* is character data (CDATA). It has a default value (#FIXED). This attribute is not provided in a document. It is automatically supplied as the value "XYZ".

Another example of element and attribute declarations is provided in Figure 10. This defines a PHOTO element in XML, so it can be used by HTML to display an image on a web page. The *src* attribute specifies the location of the photo image source file. It contains character data (CDATA) and is mandatory (#REQUIRED). The *width*, *depth*, *border* and *alt* specify the image dimensions and border thickness, as well as alternate text that is displayed while the image file is being transmitted. These are all character data (CDATA) and are optional (#IMPLIED).

```
<!ELEMENT PHOTO EMPTY>
  <!ATTLIST PHOTO src CDATA #REQUIRED>
  <!ATTLIST PHOTO width CDATA #IMPLIED>
  <!ATTLIST PHOTO depth CDATA #IMPLIED>
  <!ATTLIST PHOTO border CDATA #IMPLIED>
  <!ATTLIST PHOTO alt CDATA #IMPLIED>
```

**Figure 10:** Attribute Declarations for the PHOTO Element

### 4.3 Valid XML Documents

An XML document must not only be well formed as discussed earlier, it must also be valid. An XML document is valid if the document tags and their data content agree with the ELEMENT and ATTLIST declarations in the Document Type Definition (DTD). We discussed that a DTD is analogous to a DDL schema for a DBMS, but with different syntax. A DOCTYPE declaration for the earlier PERSON examples, together with the defined document tags and data content is shown in Figure 11.

From Figure 11, we see that a PERSON document has two attributes: a *person\_id* which must be unique (ID #REQUIRED) and sex. This is an optional attribute (#IMPLIED), but if provided it can only have the values (M | F).

A PERSON must have at least one or more *name* (name+). A *name* has zero or more *given\_name* (given\_name\*) and at least one or more *surname* (surname+). The document shows examples of these tags with the relevant data content.

A PERSON can have zero or more *email* addresses (email\*), zero or one *company*, *country* or *fax* number (company?, country?, fax?), and zero or more *phone* or *mobile* numbers (phone\*, mobile\*). We can see in Figure 11 that two phone numbers and two mobile numbers are provided as part of the PERSON document content. The data tags and content agree with the DOCTYPE declaration. The PERSON root name and its contents therefore comprise a valid XML document.

#### 4.4 Entity Declarations

XML uses the term ENTITY to declare a substitution name for insertion of predefined values. This is quite different from the use of the term “entity” in data modeling. There are two types of entity declarations. The first is a *General Entity* declaration. This can be used inside the main body of an XML document or in a DTD section, where it is called an *internal entity reference*. It can be used externally to a document, when it is called an *external entity reference*. A general entity reference is distinguished by a prefix “&”.

```
<?xml version="1.0" standalone="yes"?>
  <!DOCTYPE PERSON
  [
    <!ELEMENT PERSON
      (name+, email*, company?, country?, phone*, fax?,
mobile*)>
      <!ATTLIST PERSON person_id ID #REQUIRED>
      <!ATTLIST PERSON sex (M | F) #IMPLIED>
    <!ELEMENT name (given_name*, surname+)>
      <!ELEMENT given_name (#PCDATA)>
      <!ELEMENT surname (#PCDATA)>
    <!ELEMENT email (#PCDATA)>
    <!ELEMENT company (#PCDATA)>
    <!ELEMENT country (#PCDATA)>
```

```

<!ELEMENT phone (#PCDATA)>
<!ELEMENT fax (#PCDATA)>
]>
<PERSON person_id="p1100" sex="M">
  <person_name>
    <given_name>Clive</given_name>
    <surname>Finkelstein</surname>
  </person_name>
  <email>cfink@ies.aust.com</email>
  <company>Information Engineering Services Pty
Ltd</company>
  <country>Australia</country>
  <phone>+61-8-9402-8300</phone>
  <phone>(08) 9309-6163</phone>
  <fax>+61-8-9402-8322</fax>
  <mobile>+61-411-472-375</mobile>
  <mobile>0411-472-375</mobile>
</PERSON>

```

**Figure 11:** A Valid Internal DTD, with defined XML tags and data content

We saw internal entity references earlier, in *XML Naming Conventions*. XML supplies five predefined entities – &lt; (which is replaced by <), &gt; (by >), &amp; (by &), &quot; (by ") and &apos; (by '). The replacement text replaces an internal entity reference only when it is displayed, or is about to be processed by an application. For example, the internal entity reference "&IES;" can be declared with text "Information Engineering Services Pty Ltd". This text automatically replaces "&IES;" wherever it occurs – but only when that entity is displayed or passed to an application. Internal entities can contain references to other internal entities, but they cannot be recursive.

Distinct from an internal entity reference, the replacement text for an external entity reference immediately replaces that entity wherever it occurs. An XML parser or processor processes the replaced text as if it was an original part of the document.

An entity name must be a unique XML name. It is declared together with the replacement text. This text is substituted for the entity wherever it occurs. Figure 12 shows the internal and external format and examples for a general entity.

### Format:

```
<!ENTITY entity_name "replacement
text">
                (Internal)
```

```
<!ENTITY entity_name SYSTEM
"URL">
                (External)
```

### Declaration Examples:

```
<!ENTITY IES "Information Engineering Services Pty
Ltd">
                (Internal)
```

```
<!ENTITY copy99 "© Copyright
1999">
                (Internal)
```

```
<!ENTITY ref1 SYSTEM "http://www.ref.com/ref1.
xml">
                (External)
```

### Usage Examples:

"&IES;" is replaced later by "*Information Engineering Services Pty Ltd*"

"&copy99;" is replaced later by "© *Copyright 1999*"

"&ref1;" is replaced immediately by the content of document "*ref1.xml*"

### **Figure 12:** *General Entity Declaration and Usage Examples*

The format and two examples of an internal entity are illustrated in Figure 12. The first declares "&IES;" as an internal entity reference for "*Information Engineering Services Pty Ltd*". The second declares "&copy99;" as a shorthand for the text "© *Copyright 1999*". Whenever "&IES;" or "&copy99;" are found internally within a document they are replaced by that text, but only when the document is about to be processed or displayed.

The third example in Figure 12 declares an external entity "&ref1;" as a shorthand reference for the document "*ref1.xml*". This is located externally at "*http://www.ref-xml.com/ref1.xml*". (This is a fictitious URL.) Because it is an external entity reference, it is immediately replaced by the content of the document "*ref1.xml*".

An entity can be a convenient shorthand way of including a much larger amount of text, as shown in Figure 12. It also provides an XML document with a single point for declaration of text that can change. If volatile text appears in many places of an XML document, a general entity can be used in each place. The replacement text is defined once only when the entity is declared. Whenever that text is later changed, the updated text automatically replaces every occurrence of that entity.

We discussed that there are two types of entities. The second type is a *Parameter Entity* declaration, which can only be declared inside a DTD. A parameter entity reference is distinguished by a prefix “%”.

Figure 13 now shows the declaration format and examples of a parameter entity, which uses a prefix “%” – distinguished from general entities that use a prefix of “&”. A parameter entity is declared in a DTD, which can be internal or external. If declared in an internal DTD, it is used within that same document similar to a general entity. If declared in an external DTD, it references a URL where the DTD exists.

In the first example of Figure 13, the % character – followed by a space – declares that *person* is an external Parameter Entity. It specifies that content for “%*person*,” (no spaces) is located in the DTD file “*person.dtd*”. The content of this DTD file immediately replaces “%*person*,” as if it was an original part of the document.

### Format:

```
<!ENTITY % entity_name “replacement
text”>
           (Internal)
```

```
<!ENTITY % entity_name SYSTEM “URL”>
(External)
```

### Declaration Format:

```
<!ENTITY % person SYSTEM “person.dtd”>
(External)
```

```
<!ENTITY % idr ‘ID #REQUIRED’>
(Internal)
```

**Usage Examples:**

```

> <!DOCTYPE PERSON SYSTEM %person;
      (External)
... ..
> <!ATTLIST PERSON person_id %idr;
      (Internal)

```

**Figure 13: Parameter Entity Declaration and Usage Examples**

The second example declares “%idr;” as an internal Parameter entity that is to be immediately replaced by the text ‘ID #REQUIRED’. The example shows an ATTLIST declaration for *person\_id* (as an attribute of PERSON) with “%idr;” as an internal Parameter Entity. This is replaced immediately by “ID #REQUIRED”, as if it had been written:

```

<!ATTLIST PERSON person_id ID
#REQUIRED>

```

Any amount of replacement text can be declared for general entities and for parameter entities. This text is surrounded by quotes. As we have seen, entities can be declared to insert fragments or complete paragraphs of standard “boiler-plate” text in a document. That insertion is immediate for parameter entities or external general entities. Insertion is deferred for internal general entities; the entity is replaced by the text only when it is about to be displayed or passed to an application for processing.

**4.5 XML Resolves Many HTML Problems**

Earlier we discussed a number of problems associated with the use of HTML. XML resolves many of these problems, as summarized next. This summary also concludes the main points of the paper.

1. **XML defines content of page:** We now know that XML offers a powerful way to define tags describing the content of a document. This document can be unstructured text, or it can be graphics, images, audio or video files, or it can be structured data in legacy files, relational or object databases.
2. **Search engines can locate XML content:** Search engines can precisely locate required content based on defined XML tags. This content has more meaning than

earlier search methods that rely only on word indexes or manually defined keywords.

3. ***XML can integrate dissimilar data sources:*** XML can be used to define the structure of legacy files, relational and object databases, as well as unstructured text, graphics, images, audio or video. This makes it easier to integrate data content sourced from dissimilar systems and databases.

4. ***Easier dynamic programming:*** XML and Document Object Model (DOM) simplify programming to incorporate dynamic content from different data sources. DOM offers a language-independent interface for processing XML documents.

5. ***Easier interfacing with back-end systems:*** XML and DOM have been designed to interface with back-end systems. Special Markup languages can easily be defined for standard definition and processing of data within industries and communities with common interests based on agreed metadata definitions.

---

## 5. Author

Clive Finkelstein, acknowledged worldwide as the "Father" of Information Engineering, is Managing Director of Information Engineering Services Pty Ltd in Australia. He is the Chief Scientist of Visible Systems Corporation in the USA and is Managing Director of Visible Systems Australia Pty Ltd. He has over 45 years' experience in the Computer Industry.

This paper is based on extracts from his book: "[\*Building Corporate Portals with XML\*](#)", co-authored with Peter Aiken, published by McGraw-Hill (Sep 1999).

He has published many books and papers throughout the world including the first publication on Information Engineering: a series of six InDepth articles in US ComputerWorld in May - June 1981. He co-authored with James



Martin the influential two-volume report titled: "*Information Engineering*", published by the Savant Institute in Nov 1981. He wrote two later IE books: "[An](#)



[Introduction to Information Engineering](#)", Addison-Wesley (1989); and "[Information Engineering : Strategic Systems Development](#)", Addison-Wesley (1992). He has contributed Chapters and Forewords to books published by McGraw-Hill [["Software Engineering Productivity Handbook"](#) (1992) and Foreword: "[Data Reverse Engineering: Slaying the Legacy Dragon](#)", Peter Aiken (1996)], and by Springer-Verlag [["Handbook on Architecture of Information Systems"](#) (1998)]. His latest book is

"Enterprise Architecture for Integration: Rapid Delivery Methods and Technologies", by Clive Finkelstein, Artech House, Norwood MA (March 2006)

- [Read Book Review](#)

His current focus helps organizations to evolve from Data Warehouses and Data Marts to Corporate Portals (also called Enterprise Portals) using the Extensible Markup Language (XML). These provide a central gateway to the information and knowledge resources of an enterprise on its corporate Intranet and via the Internet. Enterprise Portal, XML and related technologies and products will rapidly become available over the next 2 – 5 years. Enterprise Portals will be the central computing focus and interface for most enterprises in the 21st century.

Clive writes a monthly column, "The Enterprise" for DM Review magazine in the USA and also publishes a free, quarterly technology newsletter via email: "The Enterprise Newsletter (TEN)". Past issues of TEN, and of the DM Review Enterprise column, are available from <http://www.ies.aust.com/articles.htm>.

---

## 6. References

1. Document Object Model (DOM) Specifications – <http://www.w3.org/TR/REC-DOM-Level-1/>
2. Finkelstein, Clive and Aiken, Peter (1999), "[\*Building Corporate Portals with XML\*](#)", McGraw-Hill, ISBN: 0-07-913705-9. Covers the design, development and deployment of Data Warehouses and Enterprise Portals using XML (560 pages).
3. Hackathorn, Richard (1998), "[\*Web Farming for the Data Warehouse\*](#)", Morgan Kaufman, ISBN: 1-55860-503-7. Includes use of XML for data sources from Internet (368 pages).
4. Harold, Elliotte Rusty (1998), "[\*XML: Extensible Markup Language\*](#)", IDG Books, ISBN: 0-7645-3199-9. Covers XML, XSL and XLL with HTML (426 pages + CD-ROM).
5. Holzner, Steven (1998), "[\*XML Complete\*](#)", McGraw-Hill, ISBN: 0-07-913702-4. Covers XML with focus on Java (516 pages + CD-ROM).
6. XML Namespaces Specifications – <http://www.w3.org/TR/REC-xml-names/>
7. Resource Description Framework (RDF) Specifications – <http://www.w3.org/Metadata/RDF/>
8. St Laurent, Simon (1998), "[\*XML: A Primer\*](#)", MIS Press [IDG Books], ISBN: 1-55828-592-X. A good basic introduction to XML (348 pages).
9. Extensible Markup Language (XML) Specifications – <http://www.w3.org/XML/>
10. <http://svc004.bne009i.server-web.com/catalogue/visible/default.shtml> lists many XML books that can be purchased directly from Amazon.com.
11. W3C: WWW Consortium and associated specifications – <http://www.w3.org/>

## 6.1 XML Information Web Sites

1. James Clark's XML Web Page – <http://www.jclark.com/xml/>
2. James Tauber's XMLINFO Web Site – <http://www.xmlinfo.com/>
3. Microsoft XML Scenarios Web Site – <http://microsoft.com/xml/scenario/intro.asp>
4. Microsoft XML Site – <http://www.microsoft.com/xml/>
5. Microsoft XML Workshop Web Site – <http://www.microsoft.com/workshop/xml/toc.htm>
6. Robin Cover's XML Resources – <http://www.sil.org/sgml/xml.html>
7. Web Farming Web Site – <http://www.webfarming.com/>
8. WWW Consortium: Specifications and Standards – <http://www.w3.org/>
9. XML.com Web Site – <http://www.xml.com/>

## 6.2 XML Development Tools: Validating Parsers

1. Data Channel's DXP Parser – <http://www.datachannel.com/products/xdk/DXP/index.html>
2. IBM's Alphaworks XML for Java Parser – <http://www.alphaworks.ibm.com/formula/xml>
3. Microsoft's MSXML Parser – <http://www.microsoft.com/standards/xml/xmlparse.htm>
4. Object Design's eXcelon XML database – <http://www.objectdesign.com/>

5. TclXML Parser (based on TCL – instead of Java or C) – <http://tcltk.anu.edu.au/XML/>

### 6.3 XML Development Tools: XML Browsers

1. Microsoft Internet Explorer 5.0 – <http://www.microsoft.com/>
2. Netscape Communicator 5.0 – <http://www.netscape.com/>
3. Netscape's Mozilla Browser – <http://www.mozilla.org/>
4. Peter Murray Rust's Jumbo Browser – <http://vsms.nottingham.ac.uk/vsms/jumbo>

| [Home](#) | [Courses](#) | [Certification](#) | [Projects](#) | [Papers](#) | [TEN Archive](#) | [EA Blog](#) |  
[Online Store](#) | [Contact Us](#) | [[Search](#)] |

(c) Copyright 2004-2006 Information Engineering Services Pty Ltd. All Rights Reserved.