

IBM XML certification success, Part 3: Review objectives of IBM XML certification

How specific XML technologies map to exam objectives

Skill Level: Introductory

[Pradeep Chopra \(authors@whizlabs.com\)](mailto:authors@whizlabs.com)

Cofounder
WHIZlabs Software

[Hari Vignesh Padmanaban \(authors@whizlabs.com\)](mailto:authors@whizlabs.com)

Software engineer
Invensys Foxboro

23 Aug 2005

Get a detailed look at the objectives of the IBM Certified Solution Developer Exam for XML and Related Technologies, and learn how specific XML technologies map to the objectives prescribed for this exam. This is the final installment of a three-part tutorial series designed specifically for those interested in XML certification. As in the first two parts of this series, authors Pradeep Chopra and Hari Vignesh Padmanaban wrap up each section with relevant examples, practice exercises, and exam tips to guide you to certification success.

Section 1. Before you start

About this tutorial

This is the final installment of a three-part tutorial series designed specifically for those interested in taking the [IBM Certified Solution Developer exam for XML and Related Technologies](#).

The first two parts examine XML technologies that are fundamental to the exam. Part 1 covers XML basics, Document Type Definitions (DTDs), W3C XML Schema, Web services, and security. Part 2 focuses on XML Path Language (XPath), Extensible Stylesheet Language Transformations (XSLT), XML Linking Language (XLink), XML Pointer Language (XPointer), Cascading Style Sheets (CSS), Extensible Stylesheet Language -- Formatting Objects (XSL-FO), the Document Object Model (DOM), and the Simple API for XML (SAX). (See [Resources](#).)

Here, the authors cover the following objectives of the certification exam in detail:

- Architecture
- Information modeling
- XML processing
- XML rendering
- XML testing and tuning

In all three tutorials, each section includes:

- An example that illustrates the theory covered in the section
- An exercise for testing your skills
- An exam tip that relates to the topics covered in the section

Note: The XML topics covered in each section of the tutorial pertain specifically to the XML certification exam and are not intended to be a comprehensive overview of XML technology.

Prerequisite: It is not required, but highly recommended that you read [Part 1](#) and [Part 2](#) before starting with this tutorial.

About the XML certification exam

The XML certification exam offered by IBM tests the user's knowledge of XML and various XML-related technologies.

The exam covers the following topics:

- XML
- Web services
- XML Path Language (XPath)

- Extensible Stylesheet Language Transformations (XSLT)
- Simple API for XML (SAX)
- The Document Object Model (DOM)
- Extensible Stylesheet Language -- Formatting Objects (XSL-FO)
- Document Type Definitions (DTDs)
- W3C XML Schema
- XML security

It is one of only a few certifications available for XML. Unlike other certifications, the XML exam requires you to not only understand these technologies, but also be able to apply them to solve real-world situations. As a result, most of the questions in the XML exam are scenario-based questions in which you are expected to choose the most appropriate solution for a given scenario.

Preparation for this exam should emphasize the importance of **practice**; anyone seriously considering XML certification should have broad experience with XML-related products *and* technologies.

For more information on the exam, visit the [information page for the IBM XML certification exam](#) where you'll find:

- Details on the **job role** and **target audience** for whom the certification was built
- **Recommended prerequisites** for the knowledge and skills one should possess before considering this certification
- The **test objectives** and the **skills** measured by the exam
- Recommended **educational resources** to prepare you for the test, based on the test objectives
- A **pre-assessment/sample test** to gauge your readiness for the actual exam

Note: *This tutorial was produced by an independent (non-IBM) organization, and is not necessarily endorsed by the IBM Certification Team.*

For general information about the IBM Professional Certification Program, visit <http://www-03.ibm.com/certify/>.

Section 2. Architecture

Introduction to Architecture

This section of the tutorial covers software **Architecture**, the first of the five exam objectives prescribed for the IBM Certified XML Solution Developer certification. It is important to understand that this section covers the architecture of the *entire* software solution being developed, including one or more XML technologies as the architectural components.

Architecture is the core of any software solution, and it is important to understand the concepts of software architecture as they apply to developing solutions using XML technologies. One of the most recent definitions of software architecture appears in the book *Software Architecture in Practice, 2nd edition* (see [Resources](#)):

The software architecture of a program or computing system is the structure or structures of the system, which comprise software elements, the externally visible properties of those elements, and the relationships among them.

Here, "externally visible properties" refers to those assumptions that other elements can make about a software element, such as its provided services, performance characteristics, fault handling, and shared resource usage.

See [Resources](#) for more information on software architecture.

When you develop software systems that use one or more XML technologies, it is important that you know the externally visible properties of the various XML technologies in order to come up with an efficient solution. The Architecture section of the certification exam tests you on this.

Now that we've defined Architecture, we'll look at the individual requirements for this exam objective:

- Determine the implications of a given architecture on XML design considerations
- Select appropriate XML technologies for an architecture
- Design functional components and interconnections for an XML application architecture
- Assess performance trade-offs related to parsing, validation, and

transformation

- Know the basics of Web services
- Address XML security using XML encryption and XML signature

Determine the implications of a given architecture on XML design considerations

A successful software solution requires that all of the individual software components be integrated in a way that enables them to work collaboratively to solve a given problem. In many cases, this might involve working with external applications, and it is important to know how all of these software components interact with each other to design a solution.

The various software components developed might use various XML technologies. This can be as simple as reading a configuration file in XML using DOM, or as complex as validating a given XML file against an XML Schema, and using XSLT for processing and rendering the XML data.

In any case, the choice of the technology needed for a component using XML has to be based on the technology of the component it will interact with. For example, it might not make sense to choose a component to use XML Schema for validating an XML file if the external legacy application from which the XML file is received supports DTD but not XML Schema. In this case, DTD is preferred over XML Schema as the technology implementation for that component.

So, while answering questions from this section, concentrate more on the problem that needs to be solved, understand the various XML technologies used (or needed) by the components involved, and know how to implement the components with the required XML technology to solve the problem being addressed

Select appropriate XML technologies for an architecture

Once you have decided on the external implications of the architecture, you must choose the appropriate XML technology to solve the problem. It is possible to use more than one XML technology to address a particular problem. For instance, to parse an XML document you could use either the DOM or SAX.

The exam tests you on the following core XML technologies:

- XML Schema 1.0
- XSLT 1.0

- DOM 2.0
- SAX 2.0
- Namespaces
- DTDs

For the exam, you should be able to:

- Choose one or more XML technologies for a specified architecture.
- Choose one from two or more XML technologies that solve a given problem.

For example, if you want to use XML for data exchange and want to make sure that the data in the file is valid, you might want to use an XML Schema for defining the data constraints rather than a DTD, as DTDs do not support strong data typing. In this case, the deciding factors might be *efficiency* and *fault handling*.

As stressed in Parts 1 and 2, the knowledge you have gained about the advantages and pitfalls of various technologies will help you answer the questions in this section.

Some of the factors that might help you choose one technology over another are:

- Performance
- Efficiency
- Ease of use
- Flexibility
- Fault handling
- Ease of implementation
- Availability of support

Design functional components and interconnections for an XML application architecture

Once you choose the appropriate XML technology for a particular architecture, you need to make decisions about the functional components that are required for using that XML technology, as well as the interconnections of these components with the other components in the solution.

For example, if you consider choosing SAX to parse an XML file, your proposed

solution should include a component that implements the SAX interface. And one of your design issues might be how that component interacts with an external application to get the XML file.

In another scenario, you might receive data from a database that you need to represent in XML. If certain fields of the database can be null, take that into consideration when you choose a schema to represent the data in XML. Even though it is possible to omit the corresponding elements (representing the fields) in your XML document, this is a bad idea that fails to convey the precise information. With XML Schema, you can represent whether the content for an element can be empty. You do this by defining the element in the Schema with the `nillable` attribute as shown here:

```
<element name="Age" nillable="true"/>
```

And then the XML document can either contain an `age` element or if it is null, can be represented as follows:

```
<age xsi:nil="true" />
```

You can answer most of the questions in this section if you know the components that are required to use the various XML technologies. You should also have a fair (if not expert) knowledge of how these components interact successfully with other components in the solution to solve a particular problem.

Note: Having knowledge of the external components used in the solution can also help you answer questions from these sections. For example, if you have to decide between CSS or XSL-FO for formatting documents, and you need to be able to render the content in different formats, you should choose XSL-FO. CSS *only* generates markup (text), whereas XSL-FO can generate binary output, representing different formats like Microsoft Word or PDF.

Assess performance trade-offs related to parsing, validation, and transformation

Performance is one of the most important criteria used to evaluate software solutions developed in the world today. With faster machines and more sophisticated tools and technologies, clients expect solutions to be fast and efficient.

Although XML is text-based and easy to understand, it can take up a lot of space. In most real-world situations, this is not a problem. For example, if you are working with a small configuration file in XML it probably won't have a very big impact on your performance when it is parsed to get information.

However, if you receive an XML file with 50,000 records from a finance-related database, XML will definitely take a big toll on the application's performance when it is parsed to extract the data (not to mention the network bandwidth needed to transfer such a huge file). If the document is parsed using DOM (thus creating a huge tree for the entire document and storing it in memory), it will make the situation worse and could easily hang up the application. For example, try opening an XML file that contains more than 3MB of data in Microsoft Internet Explorer. Internet Explorer uses DOM internally, and you can easily see the hit on performance in parsing the XML file.

In one real-world example, a company used XML files to transfer huge amounts of configuration information for their system. They realized that their network was clogged due to the huge amount of the data that transferred frequently, and it affected the performance of other applications that used the network. When they changed their XML files to Comma Separated Values (CSV), they reduced their files from megabytes to kilobytes and avoided a clogged network.

People often use technologies without evaluating the tradeoffs related to parsing, validating, and transformation of XML documents. Address these issues as early as possible. The architecture phase is the best time to do that -- the earlier, the better! A late discovery of these problems can have a huge impact on the overall architecture. In the real-time example mentioned above, even though converting files from XML to CSV worked fine, the company's programmers had to know how to write a separate CSV parser to get information from the file!

To answer questions from this section, keep in mind the tradeoffs related to the various XML technologies when you choose them for a given architecture.

Know the basics of Web services

Web services are growing increasingly popular, and are an integral part of many of the software architectures being developed to solve real-world problems. This section of the exam tests you only on the *basics* of the technologies that make up Web services, namely:

- Simple Object Access Protocol (SOAP)
- Web Services Description Language (WSDL)
- Universal Description, Discovery, and Integration (UDDI)

Web services enable you to take advantage of the functionality of existing legacy systems, and help in the integration of applications that run on different platforms and use different technologies. Many software tools on the market today offer increased support for Web services, so it is possible to develop and implement Web services in a short period of time.

It is important to keep all this in mind while you design the solution for a given problem, as it helps you to use existing functionality and also greatly reduces the time required to develop a solution.

The material covered in the [Part 1](#) of this tutorial series should help you answer the questions related to Web services in the exam.

Address XML security using XML encryption and XML signature

One of the main applications of XML is **data exchange**. Since most applications used today are XML-aware, it is possible to use XML to exchange data that's common to disparate applications implemented with different technologies. But XML is text-based, which makes it easy for anyone to read confidential data that is transferred through XML.

When you design solutions that use XML to transfer confidential information, it is important to consider the use of XML encryption and XML signature to encrypt the XML data. With these technologies, it is possible to encrypt all or part of an XML document. Encryption of XML also involves choosing a specific **encryption algorithm** and the **communication procedures** necessary to get the encryption keys.

To answer questions from this section, it is more than sufficient to know the very basics of XML security. You should be able to answer the security questions if you know the structure of the `EncryptedData` element. This is covered in [Part 1](#) of this series.

Summary for Architecture

Architecture is the backbone of every software solution that's developed, and it is important to get it right. Examine the implications of the system being developed when you consider XML technologies as part of the solution.

When more than one XML technology might be a solution to a particular problem, take into account the performance tradeoffs of the available candidate XML technologies. Web services are an integral part of many solutions being developed, and you should have a basic knowledge of this technology to consider it for a particular architecture. Also consider XML security when you design solutions that exchange confidential data with XML.

Example for Architecture

When a company wants to choose an XML technology for use by the components involved in an application, it first examines the business problem to be solved.

For example, consider the following scenario faced by Imaginary XML Test Services.

Imaginary Test Services, a leading online test provider for numerous software development technologies, decides to make test scores available to its test takers through the following media:

- Internet
- Paper
- Cell phone
- PDA

As the company develops a solution to achieve this, some important issues they must consider in deciding the data format are:

- Ability to deliver the results to multiple devices
- Storage and retrieval of the data
- Formatting the data in a way the devices expect
- Time required to develop a solution with the chosen data format

XML is the best solution for this problem.

Delivery to multiple media is one of the best uses of XML. With XML, you don't need to worry about the dissimilarity between different media when you create and maintain documents.

Also, you can maintain just one source XML file that contains the data, and create different presentations for different media on the fly. This is possible in XML because all formatting and processing is kept separate from the data.

So by choosing XML, Imaginary XML Test Services is able to solve the problem under consideration.

In addition, the time to develop the solution is also minimal since many applications and tools support XML.

Exercise for Architecture

JewelryMart is large corporation with many subsidiaries functioning independently.

To track the performance of these subsidiaries, it collects daily reports from them in the form of XML documents. The manager of JewelryMart then views these reports through a user interface. The manager can search the document and make any required changes. After this, the information is stored in a database. Which of the following technologies is the *most* appropriate for processing these XML documents?

- A. SAX
- B. DOM
- C. CSS
- D. XSL

Correct choice: B

Explanation:

SAX doesn't provide write access, so choice A is incorrect.

First, DOM parses the XML document and creates a logical representation of the information in a tree format so that an application can traverse the tree, find the relevant information, and make the required changes. For example, a search for particular information is extremely easy in this tree structure. In SAX, the document is not in memory so the application needs to maintain data structures to hold any context information.

Second, DOM provides random access, whereas in SAX an application can only handle the data in the order it arrives; therefore it can be extremely difficult to provide navigation through the document.

Third, DOM closely mirrors typical hierarchical and relational database structures -- that is, the way in which DOM represents the relationship between data elements is very similar to the way that this information is represented in modern hierarchical and relational databases. This makes it very easy to move information between a database and an XML file using DOM.

CSS and XSL are display technologies, not save/write technologies, so choices C and D are incorrect.

Exam tips for Architecture

Remember to read each question carefully! In most cases, a given question can have more than one correct choice. This means you have to draw on your

knowledge of various XML technologies, the tradeoffs associated with the technologies, the components used, and most importantly the nature of the problem that the question outlines.

Section 3. Information Modeling

Introduction to Information Modeling

This section of the tutorial covers **Information Modeling**, the second of the five objectives prescribed for the exam. This section deals with the modeling of data used by software applications. One major application of XML in software solutions that are developed today is the transfer of data between different applications. Most current Electronic Data Interchange (EDI) applications use XML.

The terms **information** and **data** relate to the same semantic definition. To avoid confusion, the term data is used as much as possible throughout this section. The W3C's DOM glossary contains the following definitions related to data modeling.

A **model** is the actual data representation for the information at hand. Examples are the structural model and the style model representing the parse structure and the style information associated with a document. The model might be a tree, or a directed graph, or something else.

A **data model** is a collection of descriptions of data structures and their contained fields, together with the operations or functions that manipulate them.

XML is used to model the data used by various applications. It is possible to model data in different ways, based on the various XML technologies available. Most of the time, the way you model data using one or more of these technologies is also determined by other factors, such as the problem domain, data integration, data, and validation.

So, it is important to come up with a good data model that takes all these factors into consideration. The **Information Modeling** section of the exam tests you on this.

Now that we've given you an overview of Information Modeling, we'll look at the individual requirements for this exam objective:

- Analyze data, documents, and problem domains
- Represent structure in XML syntax
- Use namespaces appropriately
- Define DTDs
- Define schemas using XML Schema
- Determine when to use DTD versus XML Schema
- Consider data integration and linking issues when designing data models
- Perform validation of XML documents

Analyze data, documents, and problem domains

Before you can start to model your data in XML, you must analyze the data you want to modeled in order to come up with a suitable data model. Understanding the problem domain is crucial to determining the information to be modeled.

In some cases, you might be required to model data from an already existing data source (like a document or a database). You should be able to identify the data to be modeled, depending upon the problem under discussion. For example, suppose you were to develop a data model to represent an employee's contact information from an already existing employee database. This information would be displayed in a UI, and the changes made to the information would also be written back to the database.

When you design such a data model, you first need to see which relevant information in the database is required to develop the data model. Fields like Name, Address, and Phone number might be relevant, while fields like SSN, Salary, and Employee ID might be irrelevant. If your data model is designed on the information already present in the existing data source, this will make it easier for the software components using the data model to retrieve and store the data.

Represent structure in XML syntax

Once you identify the data to be modeled, you then need to represent it in XML.

Modeling data using XML basically involves:

- Defining elements and attributes
- Splitting up data

Defining elements and attributes

All data that are required and identified are represented in XML either as **elements** or as **attributes**. It is possible to represent the same data either as an element or as an attribute. For example, you might define an employee's `id` as a separate element or as an attribute of the `employee` element. Related data are grouped as child elements under a single element, so you would group information regarding an employee (such as name and age) under a single element (`employee`).

Here are a few examples:

```
<employee>
  <name>John King</name>
  <id>se100</id>
  <age>34</age>
</employee>
```

```
<employee id="se100">
  <name>John King</name>
  <age>34</age>
</employee>
```

```
<employee id="se100">
  <name>
    <first>John</first>
    <last> King</last>
  </name>
  <age>34</age>
</employee>
```

The use of elements versus attributes has been a source of much ongoing debate. Generally, it's a good idea to use elements as much as possible to represent data. (See the developerWorks article "[When to use elements versus attributes.](#)")

Define attributes:

- For **metadata** that describes the containing element (Example of student ID: `<student id="se100">`).
- To represent **simple types** (Example of numerical units: `<length unit="cms">`)
- For data that has no frequency (is not represented more than once) or order -- such as the previous two examples, metadata and simple types

If you follow these simple rules, it is wise to define `id` as an attribute rather than as an element in the `employee` example above.

Splitting up data

In some cases, it might be necessary to divide data into two or more files. For example, if you define a data model that contains customers and orders, it is usually wise to split this data into two separate XML files that contain data related to orders and customers, respectively. That way you can pass on the customer information to applications that aren't interested in order information, and vice versa.

Use namespaces appropriately

Namespaces enable you to use a shared vocabulary. It is generally a good idea to define namespaces for elements in an XML document.

In cases where an XML document contains elements with the same names as elements in another XML file, namespaces play a major role. In this case, using namespaces enables the parser to choose the correct elements.

It is important to remember the following points when using namespaces:

- Namespaces are identified by a unique Uniform Resource Identifier (URI), so use a URI that is *unique*. A good policy is to use the URL of the relevant company Web site as the default namespace, and then define custom namespaces based on the default URI (for example, `http://www.yourCompany.com`, `http://www.yourCompany.com/Books`, or `http://www.yourCompany.com/Orders`).
- Elements might belong to the default namespace, or to the namespace of the enclosing element, even if they are not qualified names (see [Part 1](#) for more details).
- Attributes, by default, do not belong to any namespace.

Define DTDs

When you model data through XML, make sure that it follows a defined structure. You can enforce this with **Document Type Definitions (DTDs)**. You should know the constraints that you can impose on the structure of documents with DTDs.

When you design a DTD for XML documents, keep the following in mind:

- Remember the available DTD declarations:
 - DOCTYPE (defined in the XML document)
 - ELEMENT
 - ATTRIBUTE

- ENTITY
- NOTATION
- If you want to reuse a DTD, define it as a separate document (external subset) rather than in the XML document itself (internal subset).
- It is important to know the cardinality operators available in DTDs (*, +, and ?).
- How is the DTD referenced, meaning is it location dependent (SYSTEM ID) or location independent (PUBLIC ID)?
- DTDs do not provide support for namespaces.

For detailed information related to DTDs -- including declarations, syntax, advantages, and disadvantages -- please refer to [Part 1](#) (section 3) of this tutorial series.

Define schemas using XML Schema

The W3C glossary defines a **schema** as "a document that describes an XML or RDF vocabulary. Any document which describes, in formal way, a language or parameters of a language."

Besides DTDs, another way (or the most preferred way) to model your data structure in XML is by using XML Schema. XML Schemas are increasingly popular and make DTDs outdated.

To use Schemas, you need to know the following:

- You can use different declarations available in Schema to define your data structure. Besides allowing you to define attributes and elements like DTDs, Schemas also allow you to define *data types* for elements and attributes.
- Some of the important elements that you should remember from the Schema namespace are:
 - element
 - attribute
 - simpleType
 - complexType
 - Common types like string, integer, and boolean

- Remember the distinction between simple and complex types; if an element contains attributes or child elements, you must define it as a complex type.
- Know the available occurrence indicators available to model your data (`minOccurs` and `maxOccurs`)
- Know the available order indicators (`All`, `Sequence`, and `Choice`).
- Know the 12 available facets for controlling the format of the data in your XML document:
 - `enumeration`
 - `maxExclusive`
 - `minExclusive`
 - `maxInclusive`
 - `minInclusive`
 - `maxLength`
 - `minLength`
 - `pattern`
 - `length`
 - `whiteSpace`
 - `fractionDigits`
 - `totalDigits`
- Be able to form your own types (user-defined) by extending from existing simple or complex types using `<complexContent>` and `<extension>` Schema elements.
- Know that it is possible to use Schemas from multiple namespaces and other documents to construct a new Schema. This can be achieved through the `<import>` and `<include>` elements.

For more detailed information related to XML Schema elements, refer to [Part 1](#) (section 4) of this tutorial series.

This section of the exam tests your ability to construct Schemas using one or more of the above features of Schema.

Determine when to use DTD versus XML Schema

As we mentioned, you can use either DTDs or Schemas to structure your data model. Here are general rules to consider when you choose between DTDs and Schemas.

- Schemas clearly have more advantages than DTDs and *should* be the preferred technology in most cases.
- If your application has to communicate with legacy systems (sending and receiving XML documents) that understand DTDs but not Schemas, then revert to using DTDs for structuring data.
- DTDs are not XML-based, do not have namespace support (cannot have shared vocabularies), do not support DOM (you cannot modify your DTD the way you modify XML documents), and do not have strong data typing (meaning, no reliable way of enforcing correctness of data in the documents).

Consider data integration and linking issues when designing data models

When you model data for a particular application using XML and represent that data across XML documents, you can link data in one XML document to data in another document.

For example, suppose you have two documents, `Orders.xml` and `Products.xml`. `Orders.xml` contains `order` elements that contain `productID` values for the products in that order, and each `productID` corresponds to a `product` element defined in `Products.xml`.

If you have to generate a report in XML that displays all the orders along with their corresponding product descriptions, you might need to link between these two documents.

In some cases, data in one XML document might need to reference data in another XML document. In such cases, consider using **XLink** to create and describe links between XML documents.

One element can reference an element in another XML document by adding attributes from the XLink namespace. An XLink can be a **simple** or an **extended** link. An extended link allows you to link multiple resources to each other, while a simple link has just two participants -- the source and the target.

Also, if you need applications to navigate to particular parts (fragments) of your XML document, consider using XPointer. The power of XPointer lies in the fact that non-XML documents can refer to portions of XML documents!

For more details on XLink and XPointer, refer to [Part 2](#) (section 3) of this tutorial series.

Perform validation of XML documents

When you design a data model using XML, it is important to consider how the data in the XML documents will be validated. **Validation** refers to the process of ensuring that all the constraints are met.

If you were to design a data model for an employee information-related application that needs to include data about each employee's name, age, and position, then the most suitable structure for this would be:

```
<employee>
  <name>Pradeep Chopra</name>
  <age>34</age>
  <position>CEO</position>
</employee>
```

It is possible to enforce this structure for the employee data and also place constraints on the data (such as: age ≥ 18 and ≤ 60 ; the correct value for position is either "CEO" or "CTO") by using the following Schema:

```
<xs:element name="employee">
  <xs:complexType>
    <xs:all>
      <xs:element name="name" type="xs:string"/>

      <xs:element name="age">
        <xs:simpleType>
          <xs:restriction base="xs:integer">
            <xs:minInclusive value="18"/>
            <xs:maxInclusive value="60"/>
          </xs:restriction>
        </xs:simpleType>
      </xs:element>

      <xs:element name="position">
        <xs:simpleType>
          <xs:restriction base="xs:string">
            <xs:enumeration value="CTO"/>
            <xs:enumeration value="CEO"/>
          </xs:restriction>
        </xs:simpleType>
      </xs:element>
    </xs:all>
  </xs:complexType>
</xs:element>
```

It is possible for an XML document based on the employee data model to be invalid in terms of **structure** or **data**.

Invalid structure: The elements in the XML document may contain valid data for

the elements defined in the data model, but not follow the structural semantics as defined by the data model. For instance, according to the employee data model an employee should have only one age. A violation of this rule might be the presence of two `age` elements inside one `employee` element, as shown in the following example.

```
<employee>
  <name>Pradeep Chopra</name>
  <age>34</age>
  <age>34</age>
</employee>
```

Invalid data: In this case, even though the XML document might conform to the semantics of the data model, some elements of the XML document might contain invalid data. According to the employee data model, an employee can have only one age, and the valid value for age is between 18 and 60. It is invalid for an employee to have an age of 134, as shown in the example below.

```
<employee>
  <name>Pradeep Chopra</name>
  <age>134</age>
  <position>CEO</position>
</employee>
```

The designed schema ensures that the above two conditions never occur in your XML document. The `<xs:all>` indicator ensures that each `employee` element always contains *one* instance of `name`, `age`, and `position` elements. The restrictions defined for the `age` and `position` elements always ensure that the value of `age` is in the range of 0-100, and that the value of `position` is either "CEO" or "CTO".

XML technologies like XML Schema allow you to design your document to hold data in a structured format (using order and occurrence indicators) and to hold valid values for data by using constraints (using the 12 facets of XML Schema). When you are designing data models, you should be able to use these technologies to enforce the constraints required by the problem under consideration.

Summary for Information Modeling

When you represent your data model with XML, it is important to know the technologies available to model your data using DTD or Schema. Schema is the preferred mechanism, and while DTD is becoming outdated, some legacy applications still only understand DTD so take this into consideration. It is important to understand almost all of the elements defined in DTD and Schema. Knowledge of these will help you to create documents that can be validated easily and are less prone to invalid data.

Example for Information Modeling

Suppose you want to define a `name` element that satisfies the following conditions:

- Appears inside the `PersonType` element in an XML Schema
- Should not be the root element of an XML document
- Should be able to be used differently in the future -- for instance, as the `name` element inside a `BookType` element

How can you achieve this using the `xsd:complexType`, `xsd:sequence`, and `xsd:element` elements?

Refer to the definition given below:

```
<xsd:complexType name="NameType">
  <xsd:sequence>
    <xsd:element name="first" type="xsd:string"/>
    <xsd:element name="last" type="xsd:string"/>
  </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="PersonType">
  <xsd:sequence>
    <xsd:element name="name" type="NameType"/>
  </xsd:sequence>
</xsd:complexType>
```

When you define the `name` element inside `xsd:element`, which itself is a child of the `xsd:complexType` element, you prevent its use anywhere other than the `PersonType` element. Thus, `name` cannot be the root element. Also, you can define another `name` element inside a `BookType` element and have it mean whatever you want. The two `name` elements thus defined have nothing to do with each other.

Exercise for Information Modeling

An XML-based legacy application currently validates incoming XML documents with a DTD. The DTD requires frequent updates to reflect changes in the incoming XML message structure. A custom application is used to modify the message structure.

During a redesign of the system, it is proposed to remove the DTDs and instead use XML Schema to reduce the complexity and number of components used in the system. The company currently uses a custom application to modify the DTDs. Which of the following reasons is *most* likely to influence the decision to use XML Schema instead of DTD?

- A. DTDs only support weak data typing.
- B. DTDs are not extensible.
- C. DTDs have no DOM support.
- D. DTDs have limited support for namespaces.

Correct choice: C

Explanation:

Choice C is correct since DOM is a commonly used way to manipulate XML data, but it doesn't provide access to the rules of the document model in the DTD. DOM is very useful for viewing and modifying XML documents as well as their metadata. With DOM, you can modify an XML Schema as well, since an XML Schema is also a valid XML document (which a DTD is not). When you use DOM to modify the XML Schema, you can remove the custom application that modifies the DTD.

Choice A is incorrect since nowhere is it mentioned that the application requires strong data typing.

Choice B is incorrect since the application does not specify a requirement to include declarations from multiple sources.

Choice D is incorrect since namespaces cannot be used in the application. (*Tip: Do not assume any information that is not explicitly provided.*)

Exam tips for Information Modeling

In this section of the exam, you will be presented with a problem that will most likely show you the data that has to be modeled. Depending upon the problem domain, you will need to choose either a DTD or Schema for modeling that data. To answer these types of questions, you need to know the limitations and advantages of each technology with respect to the other.

Another type of question might ask you to choose one of four possible structures defined in a DTD or XML Schema for representing particular data in XML. You *should* know the syntax and options available for almost all of the elements defined in DTD and XML Schema for answering this type of questions. Not many elements are defined for DTD, so they are easy to remember. For XML Schema, the list is quite large and knowledge of the elements discussed in this tutorial series can help you answer most, if not all, of the questions in the exam.

Section 4. XML Processing

Introduction to XML Processing

This section of the tutorial covers **XML Processing**, the third of the five exam objectives prescribed for the certification exam. Here, you'll learn the various steps and technologies that software applications use to process XML documents.

The term **processing** refers to **information processing** which is best defined as *the input, verification, organization, storage, retrieval, transformation, and extraction of information from data.*

After you have modeled your data in XML, it is important to think about the various applications that process the data, and identify where in the application this information is required. You will have to provide ways for the application to access the data, and numerous ways for the application to query or modify the data.

It is possible to process XML using one or more of the following technologies: SAX, DOM, XPath, and XSLT. Each technology has its advantages and disadvantages, and it is important for you to choose the right one based on the problem being considered. Depending on your application, you might use two or more of these technologies together to solve a specific business problem.

So it is important to come up with the necessary XML technologies that take into account the above factors while processing XML -- and this is what you need to know for the XML Processing section of the exam.

Now that we've given you an overview of XML Processing, we'll look at the individual requirements for this exam objective:

- Analyze data integration points
- Use the SAX2 API to manipulate XML data
- Use the DOM2 API to manipulate XML data
- Traverse an XML document using XPath
- Transform XML using XSLT
- Determine process implications when integrating XML data
- Address validation/conformance and exception handling issues related to

XML Schema

Analyze data integration points

The Concise Tech Encyclopedia defines **data integration** as follows:

Data integration describes the process of blending data from various distinct sources to create a larger and more comprehensive data environment. One of the most important applications of data integration is the merging of legacy databases and infrastructure with current enterprise applications.

Data integration points are the parts of your application (primarily modules) that process this data. It is important to identify data integration points, which vary depending upon the problem under consideration. Some examples of data integration points might be:

- Third-party applications
- User data
- Legacy application data

These points allow you to integrate data from various sources to solve a particular problem. If the data is in XML, you have to use the appropriate XML technology to process the data. This section discusses the available XML technologies that perform XML processing.

Use the SAX 2.0 API to manipulate XML data

SAX is used to process XML documents, and in order to do this you should be familiar with the SAX 2.0 API. To answer questions related to SAX 2.0, it's important to be aware of the following *four* core interfaces defined by the SAX 2.0 API for handling events that occur during parsing:

- `ContentHandler`
- `ErrorHandler`
- `DTDHandler`
- `EntityResolver`

The exam tests you on the various methods defined by these interfaces. You need to remember the exact syntax of these methods as well as the functionality they provide. `ContentHandler` is the most important of these interfaces. A complete list

of the available methods and their functionality is described in [Part 2](#) of this tutorial series.

Apart from the syntax, it is also important to understand the working principles of SAX and the advantages of SAX over the other XML technologies that you can use to process XML documents. This can be useful as you make decisions during the design phase of a solution. For the exam, you only need to understand how SAX compares with DOM.

SAX is an *event-driven push model*, which means that when you use SAX to process XML documents, it moves in a linear fashion from the top of the document to the bottom. It does not maintain any state, but simply moves from one node to the next until it reaches the end of the document. As it goes through the document, the SAX parser fires off events that are handled by event handlers and implemented by the application using the SAX parser.

An **event handler** is an implementation of an interface provided by SAX. For example, whenever a SAX parser comes across a processing instruction in an XML document, it fires off the `processing instruction` event and the `processingInstruction(String target, String instruction)` method (event handler). This method is defined in the `ContentHandler` interface of SAX, and implemented by the software application that uses the SAX parser.

To learn more about the various interfaces that SAX provides to applications for handling events, please refer to [Part 2](#) of this tutorial series.

SAX has the following advantages:

- As it does not load the document into memory; it uses memory resources sparsely and can be used to process huge XML files.
- Since it is event driven, it can search and retrieve information very quickly from large XML documents.

But SAX also has its disadvantages:

- Since it is event driven and stateless, it cannot be used for random searches in XML documents.
- It has no built-in support for navigating between nodes.
- It cannot be used in cases where the XML documents need to be modified frequently.

Use the DOM 2.0 API to manipulate XML data

The DOM API is an alternative to SAX for processing XML documents. To use the DOM, you should be familiar with the DOM 2.0 API.

For this API, you need to be familiar with at least *three* core specifications:

- Core
- Events
- Traversal

The exam tests you on the various methods defined by these specifications. You have to remember the exact syntax of these methods as well as the functionality they provide. Core is the most important of the three specifications. A complete list of the available methods and their functionality is described in [Part 2](#) of this tutorial series.

DOM is not event driven and is a push model. When DOM is used to process an XML document, it loads the entire document into memory. The following are the advantages of DOM over SAX:

- Since the entire document is loaded into memory, it is possible to access elements randomly.
- DOM provides you with a rich set of APIs that enable you to navigate to nodes more easily than with SAX. It also has great navigation support.
- It is possible to modify elements frequently in the XML document using DOM.

But DOM also has the following disadvantages:

- Since the entire document is loaded into memory, it requires a lot of memory resources. The memory needed to construct an XML document can be nearly *10* times the size of the document! When memory is a constraint, avoid DOM as much as possible for large XML documents.
- For performing linear searches in an XML document, SAX is more efficient than DOM.

To answer questions related to SAX and DOM, you should understand the methods defined in the core interfaces and the advantages and disadvantages of these two APIs.

In many real-world applications, these technologies are used together to solve XML-related problems. In a situation where changes are made frequently to an XML document, one module in an application might process an XML document using DOM. But another module in the same application might use SAX for processing the

same XML document to display the information to a user.

Traverse an XML document using XPath

It is possible to traverse a document using XPath, the language that locates nodes (such as element, attribute, and comment nodes) in an XML document. XPath provides you with a query mechanism to get specific information from an XML document.

To answer XPath-related questions in the exam, it is imperative that you know the syntax of XPath as well as the different functions defined in XPath.

A **location path** is an expression that locates nodes in an XML document. The location path is made up of three parts:

- **Axis:** Defines the relationship between the current node and the selected nodes that would be returned by the location path.
- **Node test:** Indicates the node type that would be selected by the location path.
- **Predicate:** Allows you to filter selected nodes from the resulting node set.

XPath has 13 types of axes:

- ancestor
- ancestor-or-self
- attribute
- child
- descendant
- descendant-or-self
- following
- following-sibling
- namespace
- parent
- preceding
- preceding-sibling
- self

[Part 2](#) of this tutorial series covers each of these axes in depth, with examples.

XPath provides numerous functions that you can apply in location paths to filter out data based on test conditions and to perform calculations on specific data in the document. These functions are grouped into the following four categories.

- **Node set functions (7 functions):**

- count
- id
- last
- local-name
- name
- namespace-uri
- position

- **Number functions (5 functions):**

- ceiling
- floor
- number
- round
- sum

- **String functions (10 functions):**

- concat
- contains
- normalize-space
- starts-with
- string
- string-length
- substring
- substring-after
- substring-before
- translate

- **Boolean functions (5 functions):**

- `boolean`
- `false`
- `lang`
- `not`
- `true`

The exam tests your knowledge of these functions. Most of the time, you will be asked to either evaluate the output of an XPath expression using these functions, or select an appropriate XPath function that would solve the problem being addressed.

Again, [Part 2](#) of this tutorial series covers each of these XPath functions in depth. Other XML technologies, like XSLT and XQuery, use XPath extensively.

Transform XML using XSLT

You can extract specific information from an XML document and convert it into another XML document using XSLT. You can also convert an existing XML document into other formats, like text or HTML, using XSLT.

XSLT is often used in XML applications for transforming XML documents into other document formats, the most frequent formats being XML and HTML. To use XSLT in your applications to transform XML documents, you need to know how to write an XSLT stylesheet. An XSLT stylesheet consists of elements defined in the XSLT namespace; you should be familiar with all of these elements.

The exam tests your knowledge of these XSLT elements. A total of 35 elements are defined in the XSLT namespace. Although it is quite possible to remember the functionalities of all these elements (helping you score high on the exam!), you should, at a bare minimum, know the following frequently-used XSLT elements:

- `stylesheet`
- `template`
- `apply-templates`
- `value-of`
- `output`
- `if`
- `choose`

- when
- otherwise
- for-each
- element
- attribute
- import
- include
- variable
- call-template
- with-param
- param

Apart from the 18 important XSLT elements mentioned above, it's a good idea to know the syntax and functionalities of the remaining 17 elements.

XSLT uses XPath expressions to select nodes from an XML document. So to use XSLT, you should have a solid understanding of XPath. XSLT inherits all the functions defined in XPath, and also defines numerous additional functions that you can use to process and transform XML documents. The additional functions defined in XSLT are:

- current
- document
- element-available
- format-number
- function-available
- generate-id
- key
- system-property
- unparsed-entity-uri

It is important to understand the behavior of these various XSLT functions. [Part 2](#) of this tutorial series includes detailed explanations of all the elements and functions defined in XSLT.

Determine process implications when integrating XML data

In addition to data transfer, one of the major uses of XML is **application integration**. You can integrate various applications by converting the different data formats into XML in order to store and transfer data.

In many real-world applications, you might have to deal with existing data from legacy and external applications. These data might be in XML or some other format (such as CSV or some proprietary format). When you try to integrate data from these external applications, it is very important to assess the resulting process implications.

Some common data integration issues that might affect this process are:

- Conversion of legacy data into XML
- Conversion of XML data into other proprietary data
- Handling data errors because of integration

You'll want to consider all these issues and resolve them as early as possible (ideally during the design phase).

Address validation/conformance and exception handling issues related to XML Schema

As you saw in the previous section, "[Information modeling](#)," you can define the constraints for an XML document using XML Schema.

XML Schema supports strong data typing and namespaces, and is written in XML. For these reasons, it is the preferred choice for validating XML documents. With XML Schema, you can confirm the structure of an XML document with a pre-defined format (Schema) and also ensure that the document does not contain invalid data.

XML Schema provides support for strong data types like `string`, `float`, and `dateTime`. Strong data typing ensures that expressions are safe and data is valid. Some of the Schema elements used to define these types are: `string`, `integer`, `float`, `datetime`, `time`, and `date`. Here's an example of how to define a `string` element:

```
<xs:element name="Name" type="xs:string" />
```

Besides support for strong data types, it is also possible to restrict the values

allowed in an XML document by using XML Schema **restrictions** and **extensions**.

You can apply a restriction to simple data elements in a Schema to construct your own simple type. To do this, modify one or more facets available for the corresponding simple data type. The following example illustrates the application of the `maxLength`, `minLength`, and `pattern` facets for the `string` type to form a data type called `nameFormat`.

```
<xs:simpleType name="name">
  <xs:restriction base="xs:string">
    <xs:minLength value="5"/>
    <xs:maxLength value="10"/>
    <xs:pattern value="[a-z][A-Z]*"/>
  </xs:restriction>
</xs:simpleType>
```

W3C XML Schema defines 12 facets for simple data types. It is also possible to define complex types that might derive from other complex or simple types using the `extension` element. For a more detailed description of element types (simple and complex), extensions and restrictions, facets, and other features provided by XML Schema, please refer to [Part 1](#) of this tutorial series.

Once you define an XML Schema for an XML file, make sure that the parser that processes the XML document also supports validating the XML document by reading the corresponding Schema. Most existing XML parsers support validation of XML documents using XML Schema, and throw an exception if the XML file is invalid. Depending upon the parser, the exception thrown might vary and it is important to design the application to handle such exceptions.

Summary for XML Processing

The third section of the XML exam tests your knowledge of identifying data integration points and processing XML documents in these points using SAX 2.0 and/or DOM 2.0. It also tests your knowledge of using XPath and XSLT to transform XML documents. You should also be aware of the process implications caused by data integration, and resolve them as early as possible. In addition, you need to be able to address the usage of XML Schema for imposing constraints on XML documents, and handle exceptions that are thrown when data is invalid.

Example for XML Processing

Consider the following `books.xml` document:

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

```

<book>
  <author>abcd</author>
  <title>Understanding the Alphabet</title>
  <price>100</price>
  <pages>456</pages>
</book>
<book>
  <author>1234</author>
  <title>Understanding Numbers</title>
  <price>200</price>
  <pages>100</pages>
</book>
</books>

```

It is possible to write an XSLT stylesheet to convert the given XML document into another XML document that has only the title and price information. The structure of the desired XML document is shown in the Bookprices.xml document, below:

```

<?xml version="1.0" encoding="UTF-8"?>
<books>
<book>
  <title>Understanding the Alphabet</title>
  <price>100</price>
</book>
<book>
  <title>Understanding Numbers</title>
  <price>200</price>
</book>
</books>

```

What kind of XSLT stylesheet can achieve this? Consider the XSLT elements that are required and then look at the XSLT stylesheet below:

```

<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <!-- Specify the output to be an XML file -->
  <xsl:output method="xml" version="1.0" encoding="UTF-8"
    indent="yes"/>
  <xsl:template match="/">
    <!-- Create the books element -->
    <books>
      <!-- Call the template that matches book nodes -->
      <xsl:apply-templates select="//book"/>
    </books>
  </xsl:template>

  <!-- Template that matches the book node -->
  <xsl:template match="book">
    <!-- Create a book element -->
    <book>
      <!-- Create a title element and extract the value -->
      <title><xsl:value-of select="./title"/></title>
      <!-- Create a price element and extract the value -->
      <price><xsl:value-of select="./price"/></price>
    </book>
  </xsl:template>
</xsl:stylesheet>

```

Exercises for XML Processing

1. Which of the following are the four core interfaces of SAX 2.0?
 - A. `ContentHandler`, `ErrorHandler`, `DTDHandler`, `EntityResolver`
 - B. `ContentHandler`, `ErrorHandler`, `DTDHandler`, `EntityHandler`
 - C. `ContentHandler`, `ErrorHandler`, `DTDHandler`, `SchemaHandler`
 - D. `ContentResolver`, `ErrorHandler`, `DTDHandler`, `EntityHandler`
 - E. None of these.

Correct choice: A

Explanation:

The four interfaces of the SAX 2.0 API are `ContentHandler`, `ErrorHandler`, `DTDHandler`, and `EntityResolver`.

SAX 2.0 does not define any `EntityHandler`, `SchemaHandler` or `ContentResolver` interfaces. Thus, the remaining choices are incorrect.

2. What is the purpose of the `characters()` method defined in the SAX 2.0 API?
 - A. It is a method for inserting character data into an XML document.
 - B. It is a method of the `ContentHandler` interface for receiving notification of the character data.
 - C. It is a method of the `EntityResolver` interface for replacing an entity reference with character data.
 - D. None of the above

Correct choice: B

Explanation:

The `characters()` method provided by the `ContentHandler` interface is used to receive notification of character data.

So choices A, C, and D are incorrect.

Exam tips for XML Processing

Most of the questions from the XML Processing exam objective test you on syntax related to the technologies discussed here, so it is important to remember the API of the core interfaces of DOM and SAX. As far as XPath and XSLT are concerned, it is important to remember all of the functions defined by these technologies. Although it can be tedious to memorize the syntax of all the elements in XSLT, you should try to remember the most important ones mentioned in this section. Knowing the differences between SAX and DOM will also help you in answering some questions from the Architecture exam objective.

Section 5. XML Rendering

Introduction to XML Rendering

This section of the tutorial covers **XML Rendering**, the fourth of the five exam objectives prescribed for the exam. This section covers the various steps and technologies involved in rendering XML documents.

The term **rendering** in computer science is defined as follows: "To convert (graphics) from a file into visual form, as on a video display" (American Heritage Dictionary of the English Language, Fourth Edition).

The process of XML rendering deals with the transformation of data in XML documents into formats that are understood by different devices that use different formats for displaying data, such as browsers, printers, and other media devices.

One of the best uses of XML is its ability to deliver information to multiple media. It is possible to create different presentations for different media on the fly from a single XML document. This reduces the need to maintain different data documents for different media, and thus saves time. XML also makes the rendering process more efficient and extensible, due to the fact that the *formatting* and *processing* are kept

separate from the *data* during XML rendering.

This section deals with the ways that you can optimize such transformations, the technologies used to format the documents, the use of CSS (a non-XML technology) to format XML data, and finally the data rendering issues you need to address after you've designed the data models.

Now that we've given you an overview of XML rendering, we'll look at the individual requirements for this exam objective:

- Define optimal rendering/transformation approach for each target
- Use XSLT and Formatting Objects to style XML documents
- Re-target XML for multiple formats
- Use existing CSS to format XML or XHTML
- Consider data rendering issues after designing data models

Define optimal rendering/transformation approach for each target

Since it is possible to convert data in XML format to other formats that are target-specific, it is important to know the approach that is involved during this process.

First and foremost, you need to identify the **target applications** that receive the data. Some of the most common media that receive XML data are screens (browsers, PDAs) and print (paper).

Once you identify the targets and the required formats that the target applications expect, choose the right technology to do the transformation: XSLT transforms the data and XSL-FO formats the document. You can also use CSS to format XML documents (if the required output is markup text only).

It is crucial that you choose between the various technologies and select the appropriate approach for each target. If your target is a Web browser that requires the data to be displayed in rich formatting, then using CSS to transform the documents is a good approach. Also, Web browsers have a lot of built-in support for CSS.

On the other hand, if different clients require different parts of the same data and a lot of processing is involved, you might want to use XSLT instead of CSS. If you use XSL-FO to convert to the target format, you will need an XSL-FO processor to generate the required format.

So it is important to know the various XML technologies for rendering documents, the formats that are supported by these technologies and required by the target applications, and the applications (or libraries) needed to achieve the transformation.

Use XSLT and Formatting Objects to style XML documents

It is important to understand the subtle differences between XSLT and XSL-FO. Extensible Stylesheet Language (XSL) is a language that is used to create stylesheets that define the semantics to display an XML document. XSL is made up of the following three parts:

- *XSL Transformations (XSLT)*
- XML Path Language (XPath)
- *XSL Formatting Objects (XSL-FO)*

As you saw in [Part 2](#) of this tutorial series, XPath defines a way to address parts of XML documents and is used by XSLT. XSLT is used to transform XML documents into other XML documents or other formats (such as HTML or text). In the example for XSLT in [Part 2](#), you saw how XSLT can be applied to an XML document (`employees.xml`) to render information to a Web browser (Internet Explorer).

Even though XSLT was originally intended to perform rendering functionality, it is now primarily used to process XML documents and to generate HTML pages from XML documents. But many real-world applications require the rendering of information in XML to XSL-FO, which is used for specifying formatting semantics and for advanced styling features that are not possible with XSLT.

So, to use these XSL technologies to render XML documents, you need to have a good knowledge of both XSLT and XSL-FO. [Part 2](#) of this series covers both of these technologies to the extent required by the exam.

Re-target XML for multiple formats

As mentioned in the introduction to this section of the tutorial, the major advantage of XML is that you can deliver the same content to different media that use different formats.

For many organizations, this is one of the main reasons they choose XML. During the design phase, it is important to analyze the different formats that various clients will expect and define the necessary transformations and formatting processes that you'll define to provide those formats.

Typically, you have different XSLT stylesheets that when applied to the same XML

document produce different data. This data is then converted by a corresponding XSL-FO processor to the format expected by the client application (browser, printer, PDA, and so on).

It is also important to know that the transformation of XML is not limited to rendering it on paper or screens. Using XML, you can render data into other media formats (like voice) that are recognized by specific external applications.

Although it is possible to convert XML into different formats, you need to consider the technology for doing this (CSS or XSL-FO), and then decide on the libraries that you might need to perform the transformation to the format required by the device that receives the data. For example, if you want to display XHTML in a browser, you'll probably want to use CSS, since most existing browsers have support for CSS but few (if any) current browsers support XSL-FO.

Use existing CSS to format XML or XHTML

It is possible to use CSS to format XML or XHTML. In fact, CSS is the preferred mechanism for transforming and formatting XML documents into Web pages.

Most of the CSS-related questions in the exam test your knowledge of the advantages and disadvantages of CSS as compared to XSL-FO, so it is not really necessary for you to learn the syntax of CSS. But you should remember the following points about CSS:

- CSS has more browser support than XSL-FO.
- CSS is not based on XML and hence is not extensible.
- CSS requires a separate parser for parsing its contents.
- CSS is very simple; XSL-FO is more complex.
- Using XSL-FO, it is possible to provide features like right-to-left and footnotes; these features are not available in CSS.

Consider data rendering issues after designing data models

XML rendering is the process of displaying data stored in XML documents on devices that require it. As you have seen earlier in this section, the data might be displayed to different devices in different formats, from a single XML document.

It is important to realize that the data in an XML document is based on the data model for a specific application. For more on data modeling and the design of data models, please refer to the Information Modeling section of this tutorial.

Just as the formats of the different transformed data can vary based on the clients, so too can the data that the client displays. So it is important to consider *the data that will be rendered* and *the way the data will be rendered*.

Depending upon these two factors, you might be required to go back and redefine or change your data model. For example, you might have an XML document that contains the marks obtained by students of a particular class. If a student can view the marks on Web pages, and if the marks obtained by other students can be visible to any student, you can use CSS. However, if you want to display only the marks of a particular student, you might want to redesign your data model to store the marks separately. Or you can use XSLT to extract only the required information from the single XML document, construct a new XML document containing only the required marks, and then display it to the user through CSS.

If your application will render the data to Web pages as well as to PDAs, consider writing a separate stylesheet to convert the XML data to the required format.

Carefully analyze these design issues after the data modeling phase. This will help you identify potential rendering issues that might affect the data model and help you refine it, if necessary.

Summary for XML Rendering

XML rendering is an integral part of many software applications that use XML to store and display data. It is even more important when the data has to be rendered for multiple devices using different media formats (such as paper, screen, or voice).

XML allows you to apply transformations (using XSLT) to a single data source to extract the data to be displayed, and then apply formatting techniques (using XSL-FO) to display the data, as required. It is also possible to format XML documents using CSS. Most existing browsers support CSS, and you want to remember this when you consider XML rendering for Web pages.

Example for XML Rendering

To demonstrate the topics covered in this section, consider the following problem faced by Tech-Books-USA, Inc.

Tech-Books-USA Inc. is a leading bookseller in the United States. Apart from having numerous stores located throughout the country, it also allows customers to order books online through their Web site, tech-books.com.

An order request from the Tech-Books-USA Web server is sent as an XML message to the central server in its warehouse, where the order is fulfilled and the

XML-formatted inventory data is updated. To help the management monitor the inventory data, an XML-based application is written to generate inventory reports about the number of orders fulfilled, inventory availability, and other relevant information at the warehouse. The reports are to be available online through a variety of media, including hand-held devices.

This process works fine until the folks at Tech-Books-USA start to receive a very large number of requests, thus putting a heavy load on the server. This causes poor performance on the server.

Can you think of a way to solve this problem?

The company solves the problem by sending the client the relevant inventory data in XML, along with the stylesheet required to convert and display the data. By doing this, Tech-Books-USA shifts the process of applying the stylesheet to the XML inventory data to the client, thus reducing the workload on the server.

To answer questions in the exam, you'll want to understand the various XML technologies and be able to *implement* those technologies effectively to solve real-world problems.

Exercise for XML Rendering

Which of the following statements about XSL-FO is false?

- A. Formatting objects describe the layout of a series of nested boxes or areas that are placed on at least one page.
- B. Elements in XSL-FO objects map to the boxes on the page in a one-to-one fashion.
- C. XSL-FO allows multi-column layouts and footnotes in page margins.
- D. XSL-FO allows conditional formatting.
- E. None of the above

Correct choice: B

Explanation:

The elements in the XSL-FO document do not map to the boxes on the page in a one-to-one fashion. Instead, the XSL-FO document contains a slightly more abstract representation of the document. The formatting software uses these elements to decide which boxes to create and where to place them.

Choice A is incorrect since an XSL-FO document describes the layout of a series of nested boxes or areas. These boxes can contain text, or occasionally an external image or a horizontal rule.

Choice C is incorrect since one of the advantages of XSL-FO over CSS is that it allows multi-column layouts and footnotes in page margins. Hence, XSL-FO is a good choice for printed matter. In addition, XSL-FO allows the insertion of page numbers and automatic cross-references to particular pages by number.

Choice D is incorrect since another advantage of XSL-FO over CSS is that it allows conditional formatting based on what is actually present in a document.

Exam tips for XML Rendering

To answer questions from this section, you have to know the basics of XSLT, XSL-FO, and CSS. You should also know the syntax and working of the most important functions defined in XSLT and XSL-FO (but not CSS). It is also important to understand the differences between CSS and XSL-FO for answering most of the questions from this section.

Section 6. XML testing and tuning

Introduction to XML Testing and Tuning

This section of the tutorial covers **Testing and Tuning**, which is the last of the five objectives prescribed for the exam. This section covers the various steps and technologies involved in testing and tuning XML applications.

In *The Art of Software Testing*, Glenford Myers defines the testing of software applications as "the process of executing a program/system with the intent of finding errors."

As it applies to software, **tuning** is the process of modifying the program code of the application being developed to improve its overall efficiency.

It is important to *test* and *tune* the modules of your applications that use XML technologies.

Most testing entails ensuring that parts of the applications involved in processing XML documents work as expected. These might involve parts of your applications

that:

- Process data from legacy applications in the form of XML
- Create XML documents
- Transform XML documents
- Render XML data to other applications

With XML Schemas, you can test the validity of a data model that's represented by an XML document.

Tuning generally focuses on one or more of the following issues: execution time, memory usage, disk space, bandwidth, or some other resource. One of the most frequently encountered problems with XML is the *size* of the XML document used to store and transfer the data. For very small applications, this might not be much overhead. But for many data-centric applications, this is a serious issue as it takes more time to process the data, transform the data, and transport it across networks.

If you use certain features of XML technologies, you can fine-tune your XML applications to increase their performance and thus make them more effective. These features are discussed in this section, and they pertain mostly to those found in XSL and XML Schema. You can use these features to test and tune your XML applications.

Now that we've given you an overview of Testing and Tuning, we'll look at the individual requirements for this exam objective:

- Determine appropriateness of single versus multiple pass transformations, XSL extensions, XSL and Schema inclusions
- Optimize XML application execution
- Create instances for testing XML applications
- Refine design as necessary based on data model, data integration, data rendering, and data query issues

Determine the appropriateness of single versus multiple-pass transformations, XSL extensions, XSL and Schema inclusions

In most applications, an XML file that enters the system is processed by more than one component before it finally exits the system. In such situations, if the XML file is loaded into memory and written back by each component of the system, it can have a serious impact on the performance of the application. With multiple-pass transformations, it is possible to provide the output of one component as the input to

another.

Even if the XML data is handled by a single component and involves a huge transformation with several steps, break the transformation into separate pieces. You will have a cleaner transform that is easier to maintain and other components can reuse the individual pieces as well. For simple transformations, consider using a single transform.

It is very difficult (if not impossible) to do multiple transforms with the features of XSLT 1.0, but you can do multiple transforms using extension functions provided by various XSLT vendors. You can easily implement multiple transforms with XSLT 2.0, but the exam tests your knowledge of XSLT 1.0.

XSL extensions are a powerful feature of XSLT that call methods written in other programming languages. With `<xsl:extension>` defined in XSLT, you can include method calls from different libraries. When you use these external libraries, it's important to know the data type mappings between XSLT and the programming language that the library is written with (such as Java or C#).

It is also possible to determine the use of XSL and Schema inclusions, depending on application performance, discussed in [Optimize XML application execution](#).

Optimize XML application execution

It is very important to optimize your XML application as much as possible. You can do this to enhance an application's performance or to solve performance bottlenecks that might surface during development and testing.

One of the best ways to optimize your application is to modify the process flow to make it more effective. You can make use of specific XML semantics to optimize your XML applications. The exam tests you on the following features of XML technologies.

| <i>Table 1. XML technologies and their features</i> | |
|---|------------------------------|
| Technology | Topic |
| XML Schema | import, include, key, keyref |
| XSL | import, include, key |
| DTD | ID, IDREF |

import and include (XML Schema and XSL)

Both XML Schemas and XSLT give you the option of importing or including definitions from other schemas or stylesheets. These are great functionalities that provide a more modular design and reuse of existing definitions, thus you can avoid

duplicate code and boost application performance. This also enhances the maintenance and extensibility of applications.

The main difference between `<xsd:include>` and `<xsd:import>` in Schemas is as follows: With `<xsd:import>`, you can combine Schemas from different namespaces whereas `<xsd:include>` allows you to combine Schemas from the same namespace. When you need to create complex Schemas from different namespaces (from different applications), you can use `<xsd:import>`. `<xsd:include>` is used mainly for dividing a large Schema into smaller Schemas that can be maintained easily.

Both `<xsl:include>` and `<xsl:import>` in XSL stylesheets allow you to reuse variables, parameters, and templates defined in other stylesheets. The difference lies in the way the common templates -- templates with the same name in the original stylesheet (native templates) and the imported/included stylesheet (external templates) -- are treated. When you use `<xsl:import>`, the external templates have a *lower* priority than the native templates. By contrast, when you use `<xsl:include>`, both the native templates and the external templates have *equal* priority.

ID and IDREF (DTD)

ID and IDREF attribute types in DTDs allow you to relate different parts of the data model in an XML document. The greatest advantage of using ID and IDREF is that an XML parser validates the relationship between the data. But ID and IDREF have many limitations. Some of them are:

- Their naming rules must conform to XML grammar rules (no spaces, no slashes, and must start with a letter, underscore, or colon).
- The ID must be unique across the entire document.
- Each element can have only one ID type attribute.

key and keyref (Schema)

You can use ID/IDREF in Schema; similarly, you can use the `key` and `keyref` to associate data in XML documents. `key` and `keyref` eliminate most of the limitations that are found in DTD. Unlike DTD, `key` and `keyref` allow you to have compound keys (in DTD, an element can have only one ID typed attribute), and also allow duplicate key values. (In DTD, an ID value must be unique across the document.)

key (XSL)

`<xsl:key>` in XSLT can also define keys in the XSLT stylesheet. You can then retrieve values for a particular key using the `key()` function.

With `xsl:key` and the `key()` function, you can quickly retrieve data from XML documents.

Create instances for testing XML applications

It is always important to test your XML applications (or any other software application for that matter) to ensure that they perform as intended. You might want to develop unit tests for all your XML modules and test them as exhaustively as possible.

These tests can be as simple as creating an XML file based on a Schema, or testing an existing file based on a Schema. It might also involve testing the output of an XSLT stylesheet to ensure that it conforms to the format of the required output.

XML Schemas are a powerful way of testing XML documents. They can test the structure of an XML document as well as data validity. It is always better to validate an XML document against a Schema. If your application interacts with a legacy application that provides an XML file without a Schema, you might write your own Schema for that file based upon its structure. This can help reduce errors and can prevent you from passing invalid data to other parts of your application.

It is almost impossible to test an XSLT stylesheet with all possible combinations of real data. Instead, you should test your applications for possible values of *invalid* data and make sure that your application detects these error conditions and handles them.

Also, in cases where performance is a *requirement* (which is the case in most real-world applications developed today), make sure during the development phase that the solution you plan to implement achieves its goal. You want to ensure that ongoing development changes don't affect your application's performance. Test these types of XML applications frequently to verify that they satisfy the requirements.

It is also important to decide on performance metrics (process time, acceptable levels, and other metrics) during the testing and tuning processes. Failure to develop such metrics can lead to relative measurements, and often an exercise in futility.

Also in applications that render output to external applications in different formats, you want to test them on those external applications during development to ensure that they display the data as per the application requirements.

The exam tests your ability to create Schemas for validating XML documents and how effectively you use XSL stylesheets to transform the data.

Refine design as necessary based on data model, data integration, data rendering, and data query issues

Most of the time, you will need to redesign your XML application modules based upon your testing results or on new (or changing) customer requirements. These changes might be caused by the end results of the activities discussed in the previous three sections. In such cases, you might have to refine your data model, change your data integration process, or change the way the data is rendered. Sometimes, depending upon customer requirements, you might want to refine your design.

You can also refine the design to improve the performance of the application, to make it easier to maintain, and to extend it to accommodate new features.

Most of the time this will not involve any change in the design, so much as simply changing the way the processing is performed.

Take, for example, a Web server application that does a lot of XML processing using XSLT stylesheets. If the XSLT stylesheet differs for each client (and say you have about 500 clients) and you want to reduce the workload on the server (where the XML processing is done), you can pass the XSLT stylesheet along with the XML document to the client. Then the conversion of the document can take place on the client side. Even though you have not really changed anything in your data model, you have changed the way the processing of the documents is done to solve a problem (reduce workload on the server).

Another example might be a case where you have the results of a survey stored across several XML documents, and you have clients who frequently submit multiple queries to your application. In this case, you might want to consider storing the results in a relational database to handle all of these queries more effectively.

Summary for XML Testing and Tuning

Testing and tuning of XML applications ensures that they satisfy all the problem requirements. Testing ensures that the application works as intended while tuning improves the overall performance of the application.

To test for XML components, develop test applications (which consist of unit tests) that use instances of the XML components being developed to verify their functionality. You should do this periodically throughout the entire development process.

To achieve better performance, decide how to process the XML data (single versus multiple transforms), use other external libraries in XSL stylesheets (XSL extensions), or use the `import` and `include` functionalities provided in XSL and XML Schema. Also, you can improve an application's performance through use of `ID/IDREF` in DTDs, `key/keyref` in XML Schema, and the `key()` function in XSLT.

If you create instances and test XML applications frequently, you can ensure that your applications meet the business requirements as development takes place. Doing this also helps you to identify potential problems that might require that you redefine data or change the process flow.

Example for XML Testing and Tuning

When you create instances to test XML applications, consider several factors. For example, when you develop an XML Schema for a data model, create instances of XML documents (that contain invalid data) and test them against the given XML Schema.

Now consider the following students.xml document:

```
<students>
  <student id="100">
    <name>Hari Vignesh</name>
    <age>25</age>
    <courses>
      <course>Java Programming</course>
      <course>XML and Related Technologies</course>
    </courses>
  </student>
</students>
```

Once you develop an XML Schema to represent this structure, ensure that your Schema definition is correct. To do this, create your own XML documents and test them against the Schema. As an example, consider the following scenario:

From the data model represented by the XML file above, you see that each student can take many courses which are represented by the `course` elements. If the `course` element were defined in the Schema as:

```
<xs:element name="course" />
```

Then the XML document above would not be valid. Can you guess why?

When you define an element in an XML Schema using `<xs:element>`, "1" is the default value of `minOccurs` and `maxOccurs`. So if you want the element to occur more than once, you have to specify a value for the `maxOccurs` attribute. To fix the problem described above, include the `maxOccurs` attribute. This is shown below:

```
<xs:element name="course" maxOccurs="unbounded" />
```

When you are testing, it's good practice to:

- Use an XML document that contains real data
- Test for most probable error conditions
- Avoid testing for all possible permutations and combinations as it can be expensive

Exercise for XML Testing and Tuning

A job consulting company wants to design a system that holds the resumes and skills information for all of its employees in XML format. The system is to be designed so that users can add or modify resumes over the Internet, using XML as the intermediate message format.

The system should also allow users to query the resume information online and through hand-held devices. Users should be able to query the resume database based on the skills specified in the resume.

Which of the following resume storage techniques is *most* suited for the proposed system?

- A. Store the resumes as individual XML files in the system to facilitate easy indexing of resumes.
- B. Store all the resumes as one big XML document, so the application loads only one XML document for searches.
- C. Store the resumes in a commercially-available database that accepts XML directly and supports dynamic generation of XML data from database content.
- D. None of the above

Correct choice: C

Explanation:

If you store the resumes in a commercial database, you can use the search capabilities of the database as well as those of XML technologies like XPath and XQuery.

Choice A is incorrect because individual storage of XML files can adversely affect search performance.

Choice B is incorrect because one large XML document that contains all of the

resumes will become unmanageable. XML is not ideally suited for processing huge volumes of data from a single XML document.

Exam tips for XML Testing and Tuning

Make sure that you know the techniques that improve the performance of an XML application. Remember the differences between the `import` and `include` functions in both XSL and XML Schema. Also, you want to have a good understanding of the functionality of the `ID/IDREF` (DTDs), `key/keyref` (Schema), and `key` (XSL) functions. Read each problem carefully before you answer the question. Sometimes the best overall solution might not be the right answer for every problem. For example, if a legacy application supports only DTDs, you should go with `ID/IDREF` functions, even though `key/keyref` in Schema has many advantages over `ID/IDREF`.

Section 7. Wrap up

Summary

The third part of this tutorial series has covered the exam objectives of the [IBM Certified Solution Developer Exam for XML and Related Technologies](#) offered by IBM.

You saw how the XML technologies covered in Parts 1 and 2 (see [Resources](#)) map to the following exam objectives:

- Architecture
- Information Modeling
- XML Processing
- XML Rendering
- Testing and Tuning

To answer the questions related to **Architecture**, you should have an overall understanding of how the various XML technologies fit together to solve a given problem (DTD, DOM, Namespaces, SAX, XML Schema, and XPath in particular). You also need to know about Web services and XML security.

To answer questions related to **Information Modeling**, you need to know how to define data models using DTDs and XML Schemas.

To answer questions related to **XML Processing**, you need to know about the following technologies in detail: SAX, DOM, XPath, and XSLT.

To answer questions related to **XML Rendering**, you need to understand how CSS and XSL-FO (with XSLT) can be used for formatting XML documents.

And finally, to answer questions related to **Testing and Tuning**, you need to know how to use the specific functionalities defined in XML Schema and XSLT, and be able to test your applications and redefine data models as required.

As stressed before, it is important to understand the language (syntax) of these technologies as well as how they apply to real-world problems. In most cases, understanding the problem requirements will help you to answer the questions correctly.

The following table shows how the different XML technologies map to the exam objectives.

| | Architecture | Information Modeling | XML Processing | XML Rendering | Testing and Tuning |
|---------------------|--------------|----------------------|----------------|---------------|--------------------|
| CSS | - | - | - | Yes | - |
| DTD | Yes | Yes | - | - | Yes |
| DOM | Yes | - | Yes | - | - |
| Namespaces | Yes | - | - | - | - |
| SAX | Yes | - | Yes | - | - |
| Schema | Yes | Yes | - | - | Yes |
| Web Services | Yes | - | - | - | - |
| XLink | - | Yes | - | - | - |
| XML Security | Yes | - | - | - | - |
| XPath | Yes | - | Yes | - | - |
| XPointer | - | Yes | - | - | - |
| XSL-FO | - | - | - | Yes | - |
| XSLT | - | - | Yes | Yes | Yes |

Now that we've given you an overview of the technologies covered and the exam

objectives, you are ready to delve more deeply into these technologies. The numerous articles listed in the [Resources](#) section of this tutorial (as well as the two previous tutorials) should help you learn more about these technologies. After you have a good grasp of those, you can then take some practice tests using the exam simulators -- also listed in [Resources](#). Good luck!

Resources

Learn

- Read [Part 1](#) (March 2005) and [Part 2](#) (May 2005) of this three-part developerWorks tutorial series on preparing for the IBM XML certification exam.
- Find out more about the [IBM Certified Developer in XML and Related Technologies exam](#). This page includes:
 - Details on the job role and target audience for whom the certification was built
 - Recommended prerequisites for the knowledge and skills that one should possess before considering this certification
 - The test objectives and the skills measured by the exam
 - Recommended educational resources to prepare you for the test, based on the test objectives
 - A pre-assessment/sample test to gauge your readiness for the actual exam
- Check out this [introductory article on XML certification](#) by Pradeep Chopra, co-author of this tutorial (developerWorks, March 2003).
- Visit [the World Wide Web Consortium \(W3C\) site](#), which contains the specifications for XML and many of its related technologies. This is probably the best place to get in-depth information on XML.
- Visit www.w3schools.org for good tutorials about various XML technologies.
- Try www.zvon.org, another good site that offers tutorials on these technologies.
- Visit [PerfectXML](#) for some good tips on XSLT.
- Want to know more about software architecture? Start with [this resource page](#) from the Carnegie Mellon Software Institute.
- Read Uche Ogbuji's developerWorks article "[When to use elements versus attributes](#)" for answers to one of the oldest questions in XML design (March 2004).
- Reference the W3C's [DOM glossary](#).
- Read [Professional XML, 2nd Edition](#), the prescribed book for the XML Certification Exam that covers these technologies to the extent required by the exam.
- [Software Architecture in Practice, 2nd Edition](#) by Len Bass, Paul Clements, and

Rick Kazman is a solid reference on the topic.

- Also pick up the [XML Bible](#) by Elliotte Rusty Harold. This is a good book to get you started on XML technologies.
- [Essential XML Quick Reference](#) covers all the relevant topics except security and XSL-FO. Download a free PDF version from <http://www.develop.com/technology/developmentorseries.aspx>.
- Get [Processing XML with Java](#), a very good book for understanding SAX and DOM. Download a free online edition of this book at <http://www.cafeconleche.org/books/xmljava/>.

Get products and technologies

- Practice and assess your knowledge level using the [Whizlabs XML Exam Simulator](#).

Discuss

- Check out the [XML Certification forum](#) at JavaRanch.

About the authors

Pradeep Chopra

Pradeep Chopra is the cofounder of [Whizlabs Software](#), a global leader in IT skill assessment and certification exam preparation. A graduate of the Indian Institute of Technology (Delhi), Pradeep has consulted individuals and organizations across the globe on the values and benefits of IT certifications.

Hari Vignesh Padmanaban

Hari Vignesh Padmanaban is the author of the [Whizlabs Exam Simulator for XML](#). He has MCP, SCJP, SCWCD, SCBCD, Object Oriented Analysis and Design using UML, and XML and Related Technologies certifications to his credit. He is currently working as a software engineer for [Invensys Foxboro](#).