



**Open Applications Group**  
*Best Practices and XML Content for Everywhere-to-Everywhere Integration*

# **Practical XML Schema (PXSD)**

## **A check list for implementing XML**

**Version 1.0**

**White Paper**

**Open Applications Group, Incorporated**  
Document Number 20031021

**Author:**  
Michael Rowell  
Chief Architect  
Open Applications Group

[mrowell@openapplications.org](mailto:mrowell@openapplications.org)  
**704-630-6914**

# NOTICE

The information contained in this document is subject to change without notice.

The material in this document is published by the Open Applications Group, Inc. for evaluation. Publication of this document does not represent a commitment to implement any portion of this specification in the products of the submitters.

WHILE THE INFORMATION IN THIS PUBLICATION IS BELIEVED TO BE ACCURATE, OPEN APPLICATIONS GROUP, INC. MAKES NO WARRANTY OF ANY KIND WITH REGARD TO THIS MATERIAL INCLUDING BUT NOT LIMITED TO THE IMPLIED WARRANTIES OF MERCHANT ABILITY AND FITNESS FOR A PARTICULAR PURPOSE. Open Applications Group, Inc. shall not be liable for errors contained herein or for incidental or consequential damages in connection with the furnishing, performance or use of this material.

This document contains proprietary information, which is protected by copyright. All Rights Reserved. No part of this work covered by copyright hereon may be reproduced or used in any form or by any means—graphic, electronic, or mechanical, including photocopying, recording, taping, or information storage and retrieval systems—without permission of the copyright owner.

RESTRICTED RIGHTS LEGEND. Use, duplication, or disclosure by government is subject to restrictions as set forth in subdivision (c) (1) (ii) of the Right in Technical Data and Computer Software Clause at DFARS 252.227.7013.

*Copyright © 1995-2003 by Open Applications Group, Incorporated*

For more information, contact:  
Open Applications Group, Inc.

Telephone: 1.770.943.8364  
Internet: <http://www.openapplications.org>

# Practical XML Schema (PXSD)

## A check list for implementing XML

Version 1.0

### Table of Contents

1.	Introduction.....	5
2.	Practical list of XML Schema constructs.....	6
2.1.	The easy ones .....	6
2.1.1.	Use of named types.....	6
2.1.2.	Use of elements.....	6
2.1.3.	simpleTypes.....	6
2.1.4.	simpleType derivation by restriction .....	7
2.1.5.	complexType .....	7
2.1.6.	Attributes.....	7
2.1.7.	complexType derivation by extension.....	7
2.1.8.	Include .....	7
2.1.9.	Import .....	7
2.1.10.	Namespaces.....	7
2.1.11.	Sequence model.....	8
2.1.12.	Choice model.....	8
2.1.13.	groups.....	8
2.1.14.	Whitespace .....	8
2.1.15.	Annotations.....	8
2.1.16.	Comments .....	8
2.2.	Little more advanced Constructs .....	8
2.2.1.	Additional simpleType constructs .....	9
2.2.1.1.	Any processContents="strict" .....	9
2.2.1.2.	union of simpleTypes.....	9
2.2.2.	substitutionGroups for elements.....	9
2.2.3.	abstract Elements .....	9
2.2.4.	abstract Types .....	9
3.	Constructs to avoid .....	10
3.1.	XML Schema constructs to avoid .....	10
3.1.1.	redefine.....	10
3.1.2.	complexType derivation by restriction .....	10
3.1.3.	all model .....	10
3.2.	XML constructs to avoid.....	11
3.2.1.	xsi:type .....	11
4.	Summary .....	11

# Practical XML Schema (PXSD)

## A check list for implementing XML

Version 1.0

### ***Abstract***

*Since its introduction as a formal recommendation by the W3C of XML Schema, May 2, 2001, application vendors, integration providers and end customers have claimed that their products, and integrations support XML Schema. Many customers have learned after looking a bit deeper often times these implementations do not support the full XML Schema recommendation, and often times the level of support is merely a cursory level of support for XML Schema. The reason for this in most cases is due to the complexity of the XML Schema recommendation and the prioritization of the implementation of these features in a vendor or implementer's plans.*

*This document is provided to indicate the features and capabilities that are needed in order to practically use XML Schema in integrations today. Application vendors and integration providers that do not support the capabilities listed within this document do not practically support XML Schema.*

*This document is not intended to replace the XML Schema recommendation but rather to identify for implementers of XML and XML Schema the functionality that must/should be supported in order to claim support for XML and XML Schema. A possible extension to this document may be a compliance program that would ensure compliance to this specification.*

*The intended audience for this document are end user organizations that are using or are planning to use XML as a way to integrate their business, application vendor staff that are developing XML Schema support within their applications, and integration providers that are developing XML Schema support within their integrations, additionally, XML developers creating XML applications that are to be used for integration.*

# Practical XML Schema (PXSD)

## A check list for implementing XML

Version 1.0

### 1. Introduction

The great thing about XML is it can be used to define the structure of any information so that both the sending and receiving systems can identify the structure of the information.

This is accomplished by defining the structure that the information is to be exchanged using XML Schema (Yes, DTD's can be used as well but for the sake of this document, we will focus on XML Schema). The XML Schema definition provides the structure and the type of the data elements for both the content elements (simpleTypes) and the structured elements (complexTypees). This is accomplished by using several constructs available in the XML Schema recommendation.

The problem is that the XML Schema recommendation can be confusing and difficult to understand. Many implementers see the recommendation as ambiguous, the number of different constructs to implement daunting, and some constructs difficult (perhaps impossible) to implement in today's programming languages.

What is needed is a Practical recommendation of XML Schema that identifies the constructs that must be supported, a definitive approach to using these constructs, and a list of constructs that are to be avoided.

By doing this vendors and integrators can focus their resources on the parts of XML Schema that are practical and end customers can be reassured that their purchased applications can support this practical use of XML Schema for integration. In this way both parties can make use of standard XML applications or create there on XML applications.

This specification addresses this need by providing a practical list of XML Schema constructs that can be implemented today and identifying those constructs that at this time are difficult or impossible to implement.

#### ***Use a Standard XML Application***

By using a Standard XML Application like OAGIS it is possible to integrate your business and applications using a language that is vendor neutral and does not lock you into a single solution provider.

In today's business environment companies have to be more productive than ever, many companies are finding that they can be more productive by using a standard Canonical Business Language to integrate their heterogeneous systems. Instead of the traditional point-to-point model for integration that cost companies millions of dollars every year just to maintain.

For more information about a standard Canonical Business Language for integration please see the Open Applications Group Canonical Business Language White paper.

# Practical XML Schema (PXSD)

A check list for implementing XML

Version 1.0

## 2. Practical list of XML Schema constructs

It is important to realize that XML Schema is essentially a programming language that defines the structure of the data as it moves over the wire. In much the same way a programming language or database schema provide the structure of the data in a program, application or database.

As such these constructs should be implemented consistently across the tools and applications that intended to support them.

The following sections indicate a subset of the constructs available in XML Schema that should be supported in order to claim support for XML Schema. As such , all applications and tools that claim support for XML Schema should support these constructs.

### 2.1. *The easy ones*

First, there are some obvious constructs that must be supported in order to have the simplest support for XML Schema. These are described in the following sections.

#### 2.1.1. Use of named types

The biggest advantage of schema is it's ability define and (re)use types.

#### 2.1.2. Use of elements

You must use elements otherwise you do not have an XML document. This includes named local elements as well as global elements as well as the references to these global elements.

#### 2.1.3. simpleTypes

While XML instances carry text data XML Schema allows the schema developer to indicate what the text data fields should contain. This is the same as a program languages base types specify the type of information that a programs variables can contain.

These simpleTypes are defined by the W3C in the XML Schema recommendation, as such should be supported:

1. string
2. token
3. normalizedString
4. decimal
5. float
6. integer
7. positiveInteger
8. boolean
9. dateTime

# Practical XML Schema (PXSD)

## A check list for implementing XML

### Version 1.0

10. date
11. time
12. gYear
13. gYearMonth
14. language

#### **2.1.4. simpleType derivation by restriction**

Allows the declaration of a new simpleType that is a restriction of an existing simpleType to a subset of what is possible from the original type.

#### **2.1.5. complexType**

Allows a developer to define their own structures for use within an XML application.

#### **2.1.6. Attributes**

Allows the developer to define attributes of a given element. Attributes can only be defined as a simpleType. This says that an attribute cannot have any defined structure or further attributes.

While there has been somewhat of a religious debate in the past between attribute normal and element normal form, it is important to remember one is extensible the other is not.

#### **2.1.7. complexType derivation by extension**

Allows the extension of a base complexType in order to add addition elements or attributes.

#### **2.1.8. Include**

Allows the developer to include content from another file in the same namespace. As with any programming language for large applications it is important to structure the type of information in separate files based on their type, function, etc.

#### **2.1.9. Import**

Allows the developer to include content (elements and types) from another namespace such that they can be used within the namespace of the given file.

#### **2.1.10. Namespaces**

Allows a clear distinction of ownership/responsibility of XML applications when they are used in combination with other applications. In this manner it is possible to make use of lower level standards to build higher-level XML applications and standards.

# Practical XML Schema (PXSD)

## A check list for implementing XML

Version 1.0

### 2.1.11. Sequence model

Allows the user to identify the model in which the elements that are defined as part of a given type. The sequence model as the name implies indicates that the elements will occur in the sequence defined.

It is important to note that applications should not depend upon this exact order. Instead depend upon the hierarchical placement of elements and attributes within the structure but do not assume that a, b, and d will always occur in the given order.

### 2.1.12. Choice model

Allows the user to identify the model in which the elements that are defined as part of a given type. The choice model indicates that only one of the child elements, groups, or models may occur.

### 2.1.13. groups

Allows the developer to group commonly occurring elements together. These groups are not shown as an element with in the XML instance of the defined document but rather the elements that are contained occur as children of the element that contains the group references.

### 2.1.14. Whitespace

The schema recommendation allows white space within the XML Schemas. This white space can and should be used to make the definitions easier to read to humans. This white space should be maintained.

### 2.1.15. Annotations

Allows the developer to capture documentation and/or application information within the schema. Most systems today essentially ignore these as time goes on it will become important for applications to understand the different application information that is provided here. As long as this information is left intact at the present time

### 2.1.16. Comments

Allows the developer to comment the code.

## 2.2. *Little more advanced Constructs*

Many generally perceive these constructs to be more difficult to support at an application and integration level. As such these are the constructs that are missing in many implementations of XML Schema. This is interesting since many of these same constructs are implemented in most Object Oriented Programming languages today. As such they can be implemented in XML Schema.

# Practical XML Schema (PXSD)

## A check list for implementing XML

Version 1.0

### 2.2.1. Additional simpleType constructs

The following are additional simpleTypes that are considered more difficult to implement that must be supported.

#### 2.2.1.1. Any processContents="strict"

Allows the developer to declare that element and/or type that uses it contain any valid element as long as it is defined within the corresponding defining schema.

The processContents="strict" requires the content be defined other wise the information contained within the Any could be literally anything.

#### 2.2.1.2. union of simpleTypes

Allows a simpleType to be declared that allows the resulting type to allow contents from both of the types that were unioned together.

This approach provides a way to identify a list of enumerations while allowing the list to be extended to include other values that were not in the original list. This is the recommended approach for defining enumerations for a list of enumerations that are not fixed for a given XML application.

### 2.2.2. substitutionGroups for elements

Allow a developer to define a global element Z that is based upon a given type A. Then in the process of defining an XML application another element can be defined that either uses the same type or a type derived from A such as A' to be declared as belong to the substitutionGroup of the original global element Z lets call it global element Y. By doing this the global element Y can occur in place of the global element Z is referenced.

It is important to note that substitutionGroups cannot occur for local element only for global elements.

### 2.2.3. abstract Elements

Allows the declaration of elements that are intended to be placeholders or the head of substitution groups. This is no different than abstract variables in many programming languages.

### 2.2.4. abstract Types

Allows the declaration of abstract types that are not intended to be used directly but instead they are the base types of types that are built by extending them (possibly by restricting them.).

# Practical XML Schema (PXSD)

## A check list for implementing XML

Version 1.0

### 3. Constructs to avoid

While these constructs show promise and would be nice to have, the problem is they are not implementable in today's environment or either they are considered broken by the XML community in general.

#### 3.1. XML Schema constructs to avoid

These XML Schema constructs are considered broken or very difficult to implement.

##### 3.1.1. redefine

Allows a XML Schema developer to redefine an existing type to be something completely different. If a developer needs a new type it is easier to create a new type and use it.

The potential for the redefine is the case of using a standard XML application that allows for extensibility. In general, namespaces are used to distinguish responsibility of the XML Schema definitions this allows for extensions and modifications to the schema definitions without modifying the "standard schema definitions" by applying these modifications in a new namespace. However, since redefines do not work across namespaces this is not practical.

This construct is difficult for developers to implement.

##### 3.1.2. complexType derivation by restriction

Allows a XML Schema developer to restrict what is possible from a base type by providing a copy of the base type that the developer wants to use. Essentially, this creates a new type using the same elements and attributes of the base type that provides further restrictions on what is used in the new type. In other words it copies the original definition restricting where needed. If a new element or attribute is added to the base type, if the developer wants this available in the restricted type the developer must copy it to the restricted type.

This would be cleaner if the XML Schema recommendation had made derivation by restriction a filter where you indicate the parts that you do not want to allow to show through.

Also, like the redefine above the XML Schema Recommendation indicates that complexType derivation by restriction does not work across namespaces. Again, this is where the sort of capability is really needed to implement standard XML languages.

##### 3.1.3. all model

The all model allows the elements that are defined as children of the all model to occur in any order with in the element that contains the all model. At the end of the day when populating and/or retrieving information at a given element level it

# Practical XML Schema (PXSD)

## A check list for implementing XML

### Version 1.0

should not matter that A occurs before B. What matters is that A and B are children of C.

The all model comes close to allowing this but doesn't quite make it. If a type alpha is defined with an all model, its elements can occur in any order within an element that is of type alpha. However, if a type beta is defined that is based on alpha and its new elements are defined within an all model. Effectively, type beta has two bags that can occur in any order within those bags, but the elements of the two bags cannot be mixed.

As types are built up this can become confusing as to which elements go with which bag or all model. Because of this the all construct should not be used in order to build standard XML languages.

### 3.2. XML constructs to avoid

This refers to constructs within the XML Instance as it is transmitted over the wire that should be avoided.

#### 3.2.1. xsi:type

xsi:type allows a producer of an XML Instance to indicate on the XML Instance that instead of using the type that was specified by the XML Schema definition to use another type that is provided as a reference on the XML element as an attribute. This can be done on any element with an XML document.

Essentially, this allows the XML producer to by-pass the XML Schema type definitions provided for the document. Making every element XML Schema any element.

This construct is nearly impossible to implement in today's development environments. It also creates too many problems for an organization to be useful. For these reasons, it should be avoided.

## 4. Summary

The XML Schema recommendation was released on May 2, 2001. As organizations move to adopt XML Schema as the definition language for XML, it is critical that commercially available applications and tools implement the XML Schema recommendation consistently. Since the XML Schema recommendation is so large and yes even complex, it is important to recognize what is needed for a practical use of XML Schema.

This document identifies the constructs that are needed for a practical use of XML Schema and also identifies the constructs with XML Schema and XML that are typically avoided by mature XML Schema developers.