XML Schema 1.1 & Origo Standards

Discussion Document

Version: ***1.0 Final***

Date: ***23/08/2013***

Distribution: ***OTG***

CHANGE HISTORY

Date	Version	Change(s) Incorporated
23rd August 2013	1.0 Final	Initial and Final version of document

TABLE OF CONTENTS

1 TERMINOLOGY 4

2 INTRODUCTION 5

2.1 Background 5

2.2 Purpose of this Document 5

2.3 Structure of this Document 5

3 XML SCHEMA 1.0 6

3.1 Limitations of XML Schema 1.0 6

4 XML SCHEMA 1.1 7

4.1 Introducing XML Schema 1.1 7

4.2 Backward Compatibility 8

4.3 New Features 8

4.4 Origo Standards & XSD 1.1 8

5 SCHEMA DEPENDENCY RULES 9

5.1 What is a Schema Dependency Rule? 9

5.2 How Origo currently support Schema Dependency Rules in XSD 1.0 12

5.3 How Implementers currently support Schema Dependency Rules 13

5.4 How Schema Dependency Rules are supported with assertions in XSD 1.1 14

5.5 How Assertions in XSD 1.1 can benefit Implementers 15

5.6 How Assertions in XSD 1.1 will improve the Origo Standards 15

5.7 Recognising Schema Dependency Rule Patterns in the Origo Standards 16

6 ASSERTIONS 20

6.1 Schema Dependency Patterns and Assertions 20

6.2 Creating Assertions 20

6.3 Testing Assertions 20

7 BACKWARD COMPATIBILITY 23

8 SOFTWARE VENDORS SUPPORT 24

9 KEY DECISIONS 27

10 COSTS/BENEFITS 29

10.1 Costs 29

10.2 Benefits 30

11 APPENDIX A – XML SCHEMA 1.1 NEW FEATURES 31

12 APPENDIX B – DOCUMENTATION REFERENCES 52

1 Terminology

This section contains some of the terms and acronyms used throughout the document. Where a term or acronym is referred to in the document, a bracketed reference [n] is supplied to provide more information as and when required.

Term/Acronym	Meaning
XML	eXtensible Markup Language [1] A markup language that defines a set of rules for encoding documents in a format that is both human-readable and machine-readable.
XSD	XML Schema [2] Published as a W3C recommendation in May 2001, XSD is one of several XML Schema languages. It was the first separate schema language for XML to achieve Recommendation status by the W3C. Like all XML Schema languages, XSD can be used to express a set of rules to which an XML document must conform in order to be considered 'valid' according to that schema. However, unlike most other schema languages, XSD was also designed with the intent that determination of a document's validity would produce a collection of information adhering to specific data types.
XPath	XML Path Language [3] A query language for selecting nodes from an XML document.
Schematron	Schematron rules based validation [4] Schematron is a rule-based validation language for making assertions about the presence or absence of patterns in XML. It is a structural schema language expressed in XML using a small number of elements and XPath.
Schema Dependency Rules	A schema dependency rule describes a relationship between components in a schema which must be adhered to. Sometimes referred to as Business Rules. XSD 1.0 does not support the concept of schema dependency rules. With the introduction of assertions in XSD 1.1 these can now be supported.
Assertion	An assertion is a programmatic test which returns either true or false. XSD 1.1 provides support for assertions which can be used to support checking of schema dependency rules.
XML Data Binding	Representing information from an XML document as business objects. [5] This allows applications to access the data in the XML from the object rather than using XML parsers to retrieve the data from a direct representation of the XML itself.
XForms	An XML format for the specification of a data processing model and user interface for the XML data. XForms separates the data model from the presentation of that data by using XML to model the data and XHTML to display it. The data displayed in a form is stored in an XML document and the data submitted from the form is transported over the internet using XML. Separating data from presentation makes XForms device independent, because the data model can be used for all devices. The presentation can then be customized for different user interfaces.
QNB	Quotes and New Business Standards. These were the first Origo Standards to be published and are often referred to as the QNB Standards. They provide structures for comparison quotes and new business applications for a range of life and pensions products.
<tpsdata>	Trading Partner Specific Data. An open content structure included in the QNB Standards to allow trading partner specific data (or data structures) to be exchanged.
MIG	Message Implementation Guidelines. A document produced by Origo which describes (in non technical terms) the message structure and data content associated with a particular Origo Standard schema. This document also describes the schema dependency rules which implementers must be aware of.

2 Introduction

2.1 Background

XML Schema [2] (or XSD as it is commonly known) can be used to express a set of rules to which an XML [1] document must conform in order to be considered 'valid' according to that schema. XSD is all about expressing rules (rules about what data is allowed, how the data must be organized and the relationships between data). The primary reason for defining an XML Schema is to formally describe an XML document.

An XML Schema can be used as a contract between the sender and the receiver of an XML message: "Here's the information we agree to exchange, and the format of the information."

An additional benefit of XSD is that a schema can be used to generate code, referred to as XML Data Binding [5]. This code allows the contents of XML documents to be treated as objects within the programming environment, thus speeding up development of systems which marshal data to and from XML messages.

2.2 Purpose of this Document

XSD 1.0 was approved as a W3C Recommendation in May 2001 and has been widely adopted by a large number of XML implementations. After a long wait XSD 1.1 was approved as a W3C Recommendation in April 2012. This document looks at how the new features introduced with XSD 1.1 could benefit the Origo Standards community.

2.3 Structure of this Document

This document covers:

- The limitations of XSD 1.0;
- The new features provided in XSD 1.1;
- The new features which are relevant to the Origo Standards;
- How these new features can benefit implementers;
- The options for including support these new features in the Origo Standards;
- And the effort involved in doing so.

3 XML Schema 1.0

3.1 Limitations of XML Schema 1.0

XML Schema 1.0 is successful in that it has been widely adopted and largely achieves what it set out to. It has however been subject to some criticism:

1. It is too complicated (the specification is several hundred pages in a very technical language), so it is hard to use by non-experts — but many non-experts need schemas to describe data formats. The XSD 1.0 W3C Recommendation [6] itself is extremely difficult to read. Most users find W3Cs XML Schema Primer [7] much easier to understand.
2. There is no built in facility to support schema dependency rules.
3. There are many inconsistencies in the language, for example that elements by default are mandatory where as attributes by default are optional.

At the moment Origo Standard schemas are produced based on the XSD 1.0 specification.

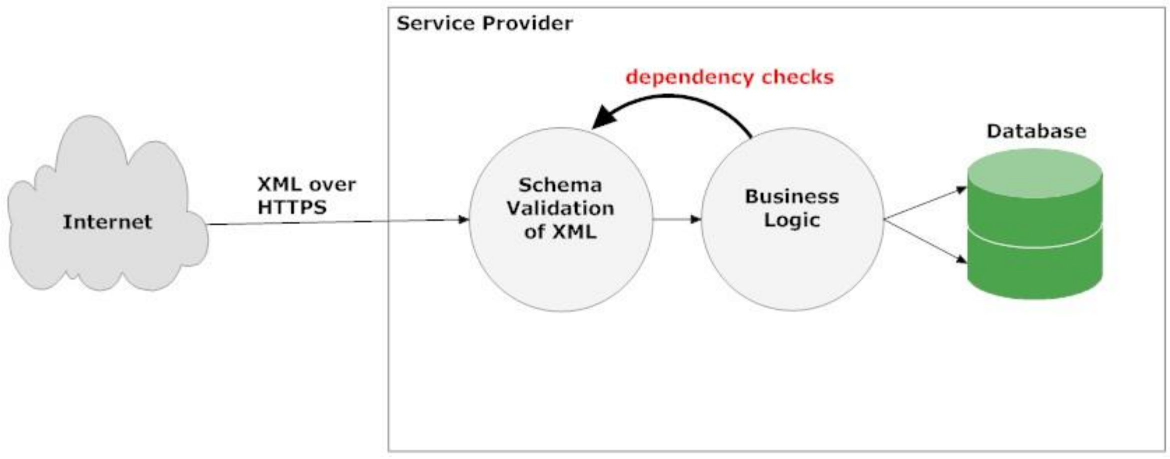
4 XML Schema 1.1

4.1 Introducing XML Schema 1.1

XSD 1.1 [8] became a W3C Recommendation in April 2012, which means it is an approved W3C specification. This in turns means that software vendors can safely implement support for the new functionality provided in XSD 1.1 without the threat that the specification will change. Indeed many software vendors have now included support for XSD 1.1.

There are a large number of improvements which have been made with the introduction of XSD 1.1 (detailed in Section), the most significant new feature is the ability to define assertions (schema dependency rule checking) against the XML document content by means of XPath 2.0 [3] expressions (an idea borrowed from Schematron [4]).

Assertions are the most useful new feature to the Origo community as this will provide the ability to carry out schema dependency rule checking at the XML validation stage, rather than including code to check this in the business logic layer. This will benefit an implementer as schema dependency rule checking will be taken care of by the existing "out of the box" XML validation process thus providing the opportunity to catch validation errors before the business logic layer.

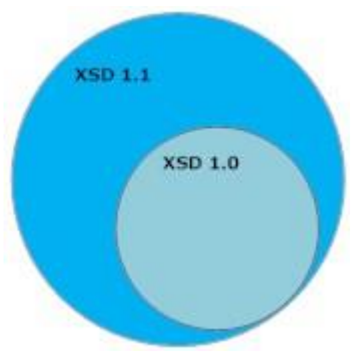


Many software vendors currently provide support for XSD 1.1 in their XML tools. (In the longer term this support could also benefit the XML Data Binding [5] process where additional functionality can be provided directly in the code generated from this process.)

At the moment there is sufficient support in the software available to provide considerable benefits for those implementers who want to take advantage of this. More details are provided later in this document, see Section 8.

4.2 Backward Compatibility

XSD 1.1 is backward compatible with XSD 1.0. This means that an XML document conforming to XSD 1.0 can be validated using the 1.1 schema validation process, but an XML document conforming to XSD 1.1 might not validate using the 1.0 validation process.



XSD 1.1 is a superset of XSD 1.0

4.3 New Features

The new features introduced in XSD 1.1 are described in Appendix A.

Each new feature is described with an associated example and its relevance to the Origo Standards.

4.4 Origo Standards & XSD 1.1

There are two new features introduced in XSD 1.1 (Section **Error! Reference source not found.**) w hich Origo rate as highly relevant to the Origo Community. These are:

1. The <assert> Element ;
2. The <assertion> Facet.

Both of these features will help Origo define schema dependency rules within the schema.

The remainder of the new features are either rated as "Medium" or "Low" relevance to the Origo Standards. This does not mean that they will not be very useful to individual implementers and how they manage schemas within their own organisation.

From an Origo Standards and Origo Community point of view the ability to process business rules (schema dependency rules) is the most useful feature in XSD 1.1. The remainder of this document will discuss schema dependency rules and the possible mechanisms for introducing XSD 1.1 automated support for these into the Origo Standards.

Note that from this point forward in the document the term "assertion" relates to the use of either the <assert> element or the <assertion> facet in XSD v1.1.

5 Schema Dependency Rules

This section will explain the following in more detail:

1. What a schema dependency rule is;
2. How Origo currently support them;
3. How implementers currently support them;
4. How they can better be supported with assertions in XSD 1.1;
5. How assertions can benefit implementers;
6. How assertions will improve the Origo Standards.

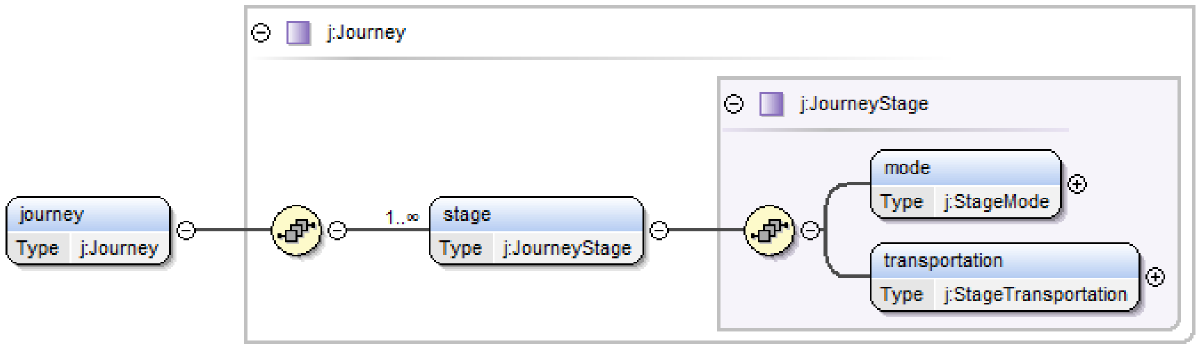
This section will also discuss the recognition of schema dependency rule patterns in the Origo Standards.

5.1 What is a Schema Dependency Rule?

A schema dependency rule is a restriction placed on the content of an XML document. This is sometimes termed a business rule or a policy although strictly speaking they are not always exactly the same thing. For example sometimes business rules or policies ***cannot*** be expressed in terms of schema components because their scope extends beyond that of a particular schema. In this case the business rule or policy isn't actually a schema dependency rule. However in the majority of cases (90%+) the business rules expressed in the documentation for the Origo Standards are actually true schema dependency rules.

Schema dependency rules are best described using an example. The small sample schema structure below describes a 'journey' with the following characteristics:

- A 'journey' must consist of one or more stages;
- Each 'stage' has a 'mode' of ***ground***, ***water*** or ***land*** travel;
- Each 'stage' also has a 'transportation' of ***airplane, bus, car***, ***hot air balloon, ship, train*** or ***yacht***.



There are some **schema dependency rules** which need to be applied to this schema:

- A mode of ***air*** must have transportation of ***airplane*** or ***hot air balloon;***
- A mode of ***ground*** must have transportation of ***bus, car*** or ***train;***
- A mode of ***water*** must have transportation of ***ship*** or ***yacht.***

i.e. "the transportation must be appropriate for the mode"

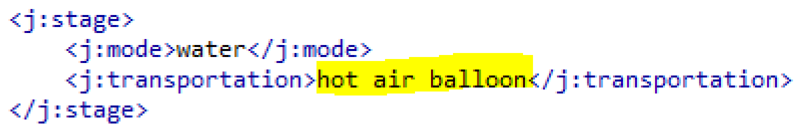
mode	allowed transportation
air	***airplane***
	hot air balloon
ground	***bus***
	car
	train
water	***ship***
	yacht

For example both snippets of XML below are **valid** in terms of XSD 1.0:

Snippet 1 is **valid** in terms of XSD 1.0

```
<j:stage>
    <j:mode>water</j:mode>
    <j:transportation>yacht</j:transportation>
</j:stage>
```

Snippet 2 is **valid** in terms of XSD 1.0 but does not meet the schema dependency rules detailed above.



```
<j:stage>
    <j:mode>water</j:mode>
    <j:transportation>hot air balloon</j:transportation>
</j:stage>
```

XSD 1.0 has no mechanism to express these rules, so this check cannot be performed at the stage where the XML document is schema validated. As a result an implementer must either:

a) write code to check for this rule using the subsequent business logic or
b) use another schema language to carry out the rule check.

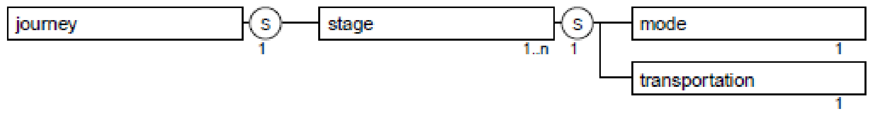
5.2 How Origo currently support Schema Dependency Rules in XSD 1.0

Origo currently express dependency rules as prose which appears in the schema itself and the Message Implementation Guidelines (MIG) for a schema. For example:

In the XSD 1.0 compliant schema there are three dependency rules which are expressed in prose.

```
<xsd:complexType name="JourneyStage">
    <xsd:sequence>
        <xsd:element name="mode" type="j:StageMode">
            <xsd:annotation>
                <xsd:documentation>Business Term: Mode of Transportation</xsd:documentation>
                <xsd:documentation>Definition: The mode of transportation for this stage of the jouney.</xsd:documentation>
            </xsd:annotation>
        </xsd:element>
        <xsd:element name="transportation" type="j:StageTransportation">
            <xsd:annotation>
                <xsd:documentation>Business Term: Transportation Mechanism</xsd:documentation>
                <xsd:documentation>Definition: The transportation mechanism by which this stage of the jouney was made.</xsd:documentation>
                <xsd:documentation>Dependency: If 'mode'="air" then this must be "airplane" or "hot air balloon".</xsd:documentation>
                <xsd:documentation>Dependency: If 'mode'="ground" then this must be "bus", "car" or "train".</xsd:documentation>
                <xsd:documentation>Dependency: If 'mode'="water" then this must be "ship" or "yacht".</xsd:documentation>
            </xsd:annotation>
        </xsd:element>
    </xsd:sequence>
</xsd:complexType>
```

In the MIG:



Name	Type	Requirements and Dependency Rules
`<j:journey>`	`j:Journey`	Business Term: Journey Definition: Used to demonstrate the specification of schema dependency rules.
`<j:stage>`	`j:JourneyStage`	Business Term: Stage of Journey Definition: Used to demonstrate the specification of schema dependency rules.
`<j:mode>`	`j:StageMode` *See j:StageMode*	Business Term: Mode of Transportation Definition: The mode of transportation for this stage of the jouney.
`<j:transportation>`	`j:StageTransportation` *See j:StageTransportation*	Business Term: Transportation Mechanism Definition: The transportation mechanism by which this stage of the jouney was made. Dependency: If 'mode'="air" then this must be "airplane" or "hot air balloon". Dependency: If 'mode'="ground" then this must be "bus", "car" or "train". Dependency: If 'mode'="water" then this must be "ship" or "yacht".

The dependency documentation provided in the XSD 1.0 compliant Origo Standards are human readable **but not** machine processable.

5.3 How Implementers currently support Schema Dependency Rules

With XSD 1.0 Origo Standards' implementers must find their own alternative way to process schema dependency rules which cannot be managed within the schema itself. There are a few ways to do this, but the two most popular mechanisms are described below.

1. Inclusion in the Implementer's Business Logic

The first mechanism is to use application code at the business logic layer to implement the schema dependency rules. A programmer must interpret the rules as specified in the documentation in the schema or MIG and translate these into their native programming language. This can require considerable effort, depending on the size and complexity of the schema. It also has the effect that these rules are not transparent - being hidden in the programming logic.

2. Use of another Schema Language

By using another schema language, for example Schematron [4], to allow the schema dependency rules to be specified in a machine readable format. This must be implemented alongside, but separate from, the usual XSD 1.0 schema validation.

A few years ago Origo investigated the use of Schematron alongside XML Schema with a view to automatically generating XForms [9] which could then be used to:

1. create Origo schema compliant messages;
2. test compliance of existing messages;
3. provide web forms to capture data related to a schema.

This was done as part of the XForms [9] research carried out by Origo in 2006-2007. XForms has never really been accepted by the industry due to lack of support in development tools and browsers. However Schematron did prove to be an ideal way to provide schema dependency rules for the Origo Standards, but because this was required in addition to XSD 1.0 there was no appetite to include Schematron files as part of the Origo Standards deliverables.

5.4 How Schema Dependency Rules are supported with assertions in XSD 1.1

Using the sample 'journey' schema as an example the schema dependency rules can be specified by using an assertion, as explained below.

The rules are as follows:

mode	allowed transportation
air	***airplane***
	hot air balloon
ground	***bus***
	car
	train
water	***ship***
	yacht

In the XSD 1.1 compliant schema we include an assertion statement at the foot of the complex type:

```
<xsd:complexType name="JourneyStage">
    <xsd:sequence>
        <xsd:element name="mode" type="j:StageMode">
            <xsd:annotation>
                <xsd:documentation>Business Term: Mode of Transportation</xsd:documentation>
                <xsd:documentation>Definition: The mode of transportation for this stage of the jouney.</xsd:documentation>
            </xsd:annotation>
        </xsd:element>
        <xsd:element name="transportation" type="j:StageTransportation">
            <xsd:annotation>
                <xsd:documentation>Business Term: Transportation Mechanism</xsd:documentation>
                <xsd:documentation>Definition: The transportation mechanism by which this stage of the jouney was made.</xsd:documentation>
                <xsd:documentation>Dependency: If 'mode'="air" then this must be "airplane" or "hot air balloon".</xsd:documentation>
                <xsd:documentation>Dependency: If 'mode'="ground" then this must be "bus", "car" or "train".</xsd:documentation>
                <xsd:documentation>Dependency: If 'mode'="water" then this must be "ship" or "yacht".</xsd:documentation>
            </xsd:annotation>
        </xsd:element>
    </xsd:sequence>
    <xsd:assert test="if (j:mode = 'air')
        then (j:transportation = 'airplane') or (j:transportation = 'hot air balloon')
        else if (j:mode = 'ground')
        then (j:transportation = 'bus') or (j:transportation = 'car') or (j:transportation = 'train')
        else if (j:mode = 'water')
        then (j:transportation = 'ship') or (j:transportation = 'yacht')
        else false()"/>
</xsd:complexType>
```

Schema validation will now **fail** if the following XML snippet was presented for validation:

```
<j:stage>
    <j:mode>water</j:mode>
    <j:transportation>hot air balloon</j:transportation>
</j:stage>
```

The XSD 1.1 assertion carries out the check which could not be handled in XSD 1.0.

NOTE: <assert> elements specified in a complex type are and'ed together, so all must be true in order that the check is passed. This can sometimes provide easier to understand assertion statements. For example the three <assert> elements below provide the same check as the single <assert> above but are simpler to read.

```
<xsd:assert
    test="if (j:mode = 'air') then (j:transportation = 'airplane') or (j:transportation = 'hot air balloon')
            else true()"/>
<xsd:assert
    test="if (j:mode = 'ground') then (j:transportation = 'bus') or (j:transportation = 'car') or (j:transportation = 'train')
            else true()"/>
<xsd:assert
    test="if (j:mode = 'water') then (j:transportation = 'ship') or (j:transportation = 'yacht')
            else true()"/>
```

5.5 How Assertions in XSD 1.1 can benefit Implementers

Implementers can benefit from the use of assertions in the following ways:

1. They allow schema dependency rules to be expressed declaratively in the schema. This keeps all rules relating to the structure and content of XML messages in one central place – the schema file;
2. They allow changes to schema dependency rules to be made easily. No complex coding changes are required;
3. They provide transparency of schema dependency rules. The centralised location of all information related to the structure and content of XML messages makes it easier to understand what is required in terms of producing valid 'business' messages.

5.6 How Assertions in XSD 1.1 will improve the Origo Standards

Similar benefits will apply to the Origo Standards, but from a slightly different perspective, as Origo do not have any implementations to maintain:

1. They allow schema dependency rules to be expressed declaratively;
2. They allow changes to schema dependency rules to be made easily (although Origo will probably maintain the prose which defines the rules in the schema documentation as well);
3. They provide transparency of schema dependency rules (although it could be said that the definition of the rules in the schema documentation, the MIG, also provides this).

NOTE: Some thought will need to be given to the impact the use of assertions may have on the Origo change control process, version management and backward compatibility rules (applicable when Origo apply minor versioning changes).

5.7 Recognising Schema Dependency Rule Patterns in the Origo Standards

In XSD 1.0 the schema dependency rules are expressed using prose, and there a certain number of 'patterns' which these rules fall into. This section explains the 'patterns' currently in use in the Origo Standards and how they can be translated to XSD 1.1 assertions.

Recognising these patterns as they are used in the Origo Standards can help with provision of the required <assert> element which will also fall into a finite set of patterns.

	Dependency Pattern	Dependency Meaning & Example
1	Required when/if ***'XPath condition'*** omitted otherwise	An optional element becomes mandatory based on a condition but must be omitted otherwise. *e.g.* <xsd:documentation>Dependency: Required if 'assessment_postponed_ind' = "true", omitted otherwise.</xsd:documentation> <xsd:documentation>Dependency: Required when 'auto_enrolment/status' is present, omitted otherwise.</xsd:documentation>
2	Required when/if ***'XPath condition'*** optional otherwise	An optional element becomes mandatory based on a condition but otherwise remains optional. *e.g.* <xsd:documentation>Dependency: Required if 'assessment_postponed_ind' = "true", otherwise optional.</xsd:documentation> <xsd:documentation>Dependency: Required when 'auto_enrolment/status' is present, otherwise optional.</xsd:documentation>
3	Optional when ***'XPath condition',*** omitted otherwise	An optional element may be provided under certain conditions but must be omitted otherwise. *e.g.* <xsd:documentation>Dependency: Optional if 'auto_enrolment/status' = "Non-Eligible Job Holder" OR

	Dependency Pattern	Dependency Meaning & Example
		"Entitled Worker", omitted otherwise.</xsd:documentation> <xsd:documentation>Dependency: Optional if 'opt_in_date' is present, omitted otherwise.</xsd:documentation>
4	Omitted when ***'XPath condition'***, optional otherwise	An optional element is not required under certain conditions, but remains optional otherwise. *e.g.* <xsd:documentation>Dependency: Omitted if 'opt_out_date' is present and 'opt_out_date_valid_ind' = "true", optional otherwise.</xsd:documentation>
5	At least one of the child elements must be present if the parent element is present	At least one of the optional child elements must be present if the optional parent is present. *e.g.* <xsd:documentation>Dependency: At least one of the child elements must be present if the parent element is present.</xsd:documentation>
6	At least one of the child elements must be present	At least one of the optional child elements must be present in a mandatory parent element. *e.g.* <xsd:documentation>Dependency: At least one of the child elements must be present.</xsd:documentation>
7	Either ***'XPath condition 1'*** or ***'XPath condition 2'*** but not both	Distinct optional elements are not permitted together in the message. Either one or the other can optionally be present but not both. *e.g.* <xsd:documentation>Dependency: Agency details may appear either as a Group/Sub-Group, or against the contract, but not both.</xsd:documentation>
8	If ***'XPath condition'*** then at least one of <list> must be present	Under certain conditions at least one of a list of elements must be present. *e.g.* <xsd:documentation>Dependency: If 'dimensions' is present, at least one of 'region_dimension',

	Dependency Pattern	Dependency Meaning & Example
		‘product_line_dimension’ and ‘tps_dimensions’ must be present.</xsd:documentation>
9	At least one of <list of elements> must be present	At least one of a list of optional elements must be present. *e.g.* <xsd:documentation>Dependency: At least one of ‘item/reference’ and ‘item/description’ must be present.</xsd:documentation>

Each of the schema dependency patterns described above can be represented by an <assertion> using XPath 2.0.

For example taking the first dependency pattern above to show the resultant assertions which are needed to automate the checking of the schema dependency rules.

	Dependency Pattern	Dependency Meaning & Example
1	Required when/if ***‘XPath condition’*** omitted otherwise	An optional element becomes mandatory based on a condition but must be omitted otherwise. *e.g. on the ‘assessment_date’ element in the Auto Enrolment Employee List schema.* <xsd:documentation>Dependency: Required when ‘auto_enrolment/status’ is present, omitted otherwise.</xsd:documentation> **<!-- Make sure if status is supplied then the assessment_date is also supplied -->** **<xsd:assert test="if (status) then assessment_date** **else if (not(status))** **then true()** **else false()"/>**

	Dependency Pattern	Dependency Meaning & Example
		e.g. on 'reassessment_date' element in the Auto Enrolment Employee List schema. <xsd:documentation>Dependency: Required if 'assessment_postponed_ind' = "true", omitted otherwise.</xsd:documentation> **<!-- Make sure if assessment_postponed_ind is true then the reassessment_date is provided, but must be omitted otherwise -->** **<xsd:assert test="if (assessment_postponed_ind = 'true') then reassessment_date** **else if (not(assessment_postponed_ind))** **then not(reassessment_date)** **else if (assessment_postponed_ind = 'false')** **then not(reassessment_date)** **else false()"/>**

Using the schema dependency rule pattern approach it would be possible to document a set of 'skeletal' assertions which could be used as a starting point for creating any assertions which would be required in the future. This would speed up the otherwise fairly complex and time consuming process for creating all the assertions required for each schema.

6 Assertions

This section describes in more detail the technicalities of creating assertions and the testing effort involved in producing assertions.

6.1 Schema Dependency Patterns and Assertions

Assertions are essentially logical statements which return a value of true or false. An assertion will check that a schema dependency rule is being complied with (or not). In XSD 1.1 assertions can be checked at the schema validation processing stage. Assertions must be authored as part of the schema development and uses XPath 2.0 to provide reference to any required schema components.

6.2 Creating Assertions

Assertion statements can be built based on the schema dependency patterns described in Section 5.7.

Origo recommend specifying several <assert> elements instead of one large complex <assert> element.

For example:-

Using the example in section 5.7 where the schema dependency rule is "*Make sure if 'assessment_postponed_ind' is true then the 'reassessment_date' is provided, but must be omitted otherwise* "

The single <assert> element could be specified as

```
<xsd:assert test="if (assessment_postponed_ind = 'true') then reassessment_date
                        else if (not(assessment_postponed_ind))
                        then not(reassessment_date)
                        else if (assessment_postponed_ind = 'false')
                        then not(reassessment_date)
                        else false()"/>
```

This is a fairly complex statement, which would be simpler to understand if it was split across three assertions (which will be and'ed together) as follows:

```
<xsd:assert test="if (assessment_postponed_ind = 'true') then reassessment_date else true()"/>
<xsd:assert test="if (not(assessment_postponed_ind)) then not(reassessment_date) else true()"/>
<xsd:assert test="if (assessment_postponed_ind = 'false') then not(reassessment_date) else true()"/>
```

It is advantageous to keep <assert> elements as simple to understand as possible.

6.3 Testing Assertions

Assertions require to be tested to ensure that they cover the requirements.

The most effective approach to testing assertions is to use example instance documents which represent the various combinations of data which an assertion is expected to manage.

For example:-

Using the example in section 5.7 where the schema dependency rule is *"Make sure if 'assessment_postponed_ind' is true then the 'reassessment_date' is provided, but must be omitted otherwise "*

which is represented by the <assert> elements:

A1: <xsd:assert test="if (assessment_postponed_ind = 'true') then reassessment_date else true()"/>
A2: <xsd:assert test="if (not(assessment_postponed_ind)) then not(reassessment_date) else true()"/>
A3: <xsd:assert test="if (assessment_postponed_ind = 'false') then not(reassessment_date) else true()"/>

The following XML snippets can be used to test these <assert> elements:

XML Example	**Schema Validation Result**

<automatic_enrolment>
    <qualifying_earnings>
        <amount currency="GBP">25000.00</amount>
        <period>Annual</period>
    </qualifying_earnings>
    <status>Eligible Job Holder</status>
    <status_decision>Auto Enrolment</status_decision>
    <assessment_date>2012-08-18</assessment_date>
    <assessment_postponed_ind>true</assessment_postponed_ind>
    <reassessment_date>2013-08-18</reassessment_date>
</automatic_enrolment>
	Pass
	
	A1, **A2** and **A3** all true

<automatic_enrolment>
    <qualifying_earnings>
        <amount currency="GBP">25000.00</amount>
        <period>Annual</period>
    </qualifying_earnings>
    <status>Eligible Job Holder</status>
    <status_decision>Auto Enrolment</status_decision>
    <assessment_date>2012-08-18</assessment_date>
    <assessment_postponed_ind>false</assessment_postponed_ind>
    <reassessment_date>2013-08-18</reassessment_date>
</automatic_enrolment>
	Fail
	
	'reassessment_date' is not required because the assessment has not been postponed.
	
	A3 false

<automatic_enrolment>
    <qualifying_earnings>
        <amount currency="GBP">25000.00</amount>
	Fail
	
	'reassessment_date' is required because

XML Example	Schema Validation Result
<period>Annual</period> </qualifying_earnings> <status>Eligible Job Holder</status> <status_decision>Auto Enrolment</status_decision> <assessment_date>2012-08-18</assessment_date> **<assessment_postponed_ind>true</assessment_postponed_ind>** </automatic_enrolment>	assessment has been postponed. **A1** false
<automatic_enrolment> <qualifying_earnings> <amount currency="GBP">25000.00</amount> <period>Annual</period> </qualifying_earnings> <status>Eligible Job Holder</status> <status_decision>Auto Enrolment</status_decision> <assessment_date>2012-08-18</assessment_date> **<reassessment_date>2013-08-18</reassessment_date>** </automatic_enrolment>	Fail ‘reassessment_date’ is not required because the assessment has not been postponed. **A2** false

Breaking one large <assert> element into a number of smaller <assert> elements also makes testing much more manageable.

7 Backward Compatibility

This section describes how Origo could provide support for both XSD 1.1 and XSD 1.0 (allowing those not able to support XSD 1.1 to still effectively use the Standards).

Assuming that Origo will only provide support for assertions (schema dependency rules) there are two ways this could be achieved:

Approach	Impact
Origo would publish both XSD 1.0 and XSD 1.1 versions of all schemas. The XSD 1.0 version will not include any <assert> elements or <assertion> facets. The XSD 1.1 version will only differ from the XSD 1.0 version by the addition of <assert> elements and <assertion> facets.	**Implementers** Implementers need to make the choice which version of the schema they need. **Origo** Origo would need to publish two versions of each schema. Only one version would be maintained by Origo with a suitable process applied to generate the second from the first.
Origo would only publish the XSD 1.1 version of all schemas which would include <assert> elements and <assertion> facets. A transformation process (XSLT) would be made available to allow implementers to remove the <assert> elements and <assertion> facets where they cannot be supported.	**Implementers** Implementers need to remove the assertions from the schemas using the published XSLT transformation if their XML processing capabilities cannot understand XSD 1.1. **Origo** Origo would maintain and publish one version of each schema. Origo would also need to supply a reliable mechanism for removing assertions where they are not wanted or needed.

Based on an agreement to proceed with XSD 1.1, the option to choose for delivery of support for assertions is one of the key decisions which must be made; see Section 9 for more details.

8 Software Vendors Support

This section provides details of:

1. Which XML tools supports XML Schema 1.1;
2. How much of the functionality is supported;
3. When support is planned if not already provided;
4. Suitable alternatives if support is not available.

Toolset	XSD 1.1 features supported	Notes
Saxon [11] Saxonica software providing XSD, XSLT and XQuery processing capabilities.	Full support	V9.4 supports all features of XSD 1.1 V9.1 introduced initial support for major features of XSD 1.1
XERCES [12] Apache software used for parsing, validating, serializing and manipulating XML.	Majority support	V2.11 supports the majority of features of XSD 1.1 (full support for assertions is included)
JAXP [13] A Java XML API which provides the capability to	Majority support	V1.4 supports the majority of features of XSD 1.1 (and forms part of the standard Java Development Kit) and provides many of the APIs used by many other XML tools including Saxon and XERCES. Full support for assertions is included.

Toolset	XSD 1.1 features supported	Notes
parse, validate and manipulate XML.		
Oxygen XML Editor [14] XML editor and authoring tool which provides facilities to validate XML and debug XSLT and XQuery.	Full support	Oxygen offers full compliance with both XSD 1.0 and 1.1. This is the tool preferred by Origo for development of the Origo Standards.
Microsoft .NET Framework [15]	No support	The .NET framework provides a number of facilities for working with XML including MSXML, XmlSerialiser and Visual Studio. None of these support XSD 1.1 at the moment and there are no publicised plans to do so in the future. There is a .NET compatible release of Saxon called Saxon .NET which would provide an alternative for .NET implementers who wish to process XSD 1.1 schemas.
Liquid XML [16] Liquid XML studio integrates into Microsoft Visual Studio allowing development of XML and XSD applications.	No support	Liquid XML integrates with the .NET framework but has no support for XSD 1.1 and no publicised plans to do so in the future. There is a .NET compatible release of Saxon called Saxon .NET which would provide an alternative for .NET implementers who wish to process XSD 1.1 schemas.

Toolset	XSD 1.1 features supported	Notes
Altova XMLSpy [17] XML editor and authoring tool which provides facilities to validate XML and debug XSLT and XQuery.	No support	XMLSpy offers compliance with the majority of the XSD 1.0 specification, but there is no current support for XSD 1.1. Altova have indicated that they do plan to provide support in a future release. Oxygen can be used as an alternative to XMLSpy.

The table above is not an exhaustive list of XML tools but represents some of the most commonly used.

9 Key Decisions

This discussion document is intended to provoke thought on the subject of XSD 1.1 support in the Origo Standards.

Key Decision Required	Notes
Should assertions be included in the Origo Standards as part of a published schema?	There is sufficient support for XSD 1.1 in the XML tools available that some implementers could achieve considerable benefit from this feature.
Should assertions be included in new Origo Standards only or should they be added to existing Origo Standards?	To take full advantage of assertions they should be applied to all Origo Standards. However it would probably make sense to identify one particular standard to address first and assess the development and testing effort involved in adding assertions.
Should assertions be added to existing Origo Standards on an ongoing maintenance basis or by a concerted project based effort?	This depends on the amount of effort involved in including support for assertions in any particular Origo Standard. If it's minimal then it may be better to do this on an ongoing maintenance basis. If it involves considerable effort then perhaps a project basis may be more appropriate.
Should schema annotations (which describe the schema dependency rules to be processed by the assertions) remain as part of the published schema?	The annotations will still be required in order to satisfy the requirements for generation of the MIGs.
The options for publication of the Origo Standards which support XSD 1.1 are discussed in Section 7 where backward compatibility is a key issue. Origo can either: 1. Publish XSD 1.1 compliant version of the Origo Standards and allow implementers to drop assertions if they don't need them; 2. Or publish both XSD 1.0 and 1.1 compliant versions of the Origo	There are pros and cons to both options available, see Section 7 for details.

Key Decision Required	Notes
Standards with implementers making the appropriate choice.	

10 Costs/Benefits

This section provides details of the associated costs and benefits of providing support for assertions in the Origo Standards.

It should be noted that the costs will only be incurred once by Origo during the process of creating and testing the assertions, whereas the benefits will be gained by each implementing organisation.

10.1 Costs

The costs (in terms of Origo resource) involved in developing support for assertions in an Origo Standard can be summarised as follows:

- the development and testing effort required in creating the <assert> elements and <assertion> facets associated with the annotated schema dependency rules in each schema;
- the development of any style sheets necessary to transform schemas to support backward compatibility requirements;
- Updating the processes involved in the internal schema development lifecycle.

Schema Development

Task required	Notes
Create <assert> element	Using the dependency rule patterns documented in Section 5.7 it should be possible to quickly identify a pattern to be used to model the <assert> element required. The <assert> element may need to be split into several <assert> elements for simplicity (see Section 6.2).
Test <assert> element	Testing an assertion will require each potential combination of values to be provided in sample XML messages with each being schema validated during the test (see Section 6.3) This could be quite time consuming and is dependent on the complexity of the schema dependency rule being modelled as an assertion.
Create <assertion> facet	<assertion> facets are much simpler to create than <assert> elements. They will also be used much more infrequently than <assertion> elements.

Task required	Notes
Test <assertion> facet	Similarly it should be simpler to test an <assertion> facet.

Transformation Processes

There may be the need to publish both XSD 1.0 and 1.1 versions of the Origo schemas. In this case it is most likely that Origo will develop one master copy of the schema (1.1) and use a transformation process to generate the other version (1.0) ready for publication. The transformation process requires to be developed as part of the support for XSD 1.1.

Internal Schema Development Lifecycle Processes

The internal processes used by Origo as part of normal schema development life cycle will need to be adapted to accommodate XSD 1.1 and assertions. These include:

- updating internal document generation processes (MIG generation);
- updating internal schema quality assurance processes.

Using the 'off the shelf' support for XSD 1.1 provided by XML parsers should keep costs low for implementers.

10.2 Benefits

The benefits to implementers can be summarised as the inclusion of assertions in the schemas allowing dependency checks to be invoked as part of the schema validation stage of XML message processing. Without assertions the onus is on the implementer to provide some way of performing these dependency checks.

In many circumstances these checks are only performed as part of the programming logic in the business layer of the application handling the data being passed via the XML message. By managing these checks at the schema validation stage, invalid XML content should not reach the business layer of the application.

11 Appendix A – XML Schema 1.1 New Features

Feature	Description & Example	Relevance to Origo Schemas

The <assert> Element	Policies and rules are expressed in XSD 1.1 using assertions, *e.g.*
	
	<[1]element name="publication" type="PublicationType"/>
	
	<complexType name="PublicationType">
	    < sequence>
	        < element name="title" type="string" />
	        < element name="author" type="string" />
	        < element name="date" type="string" />
	        < element name="isbn" type="string" minOccurs="0"/>
	        < element name="publisher" type="string" minOccurs="0"/>
	    </ sequence>
	    < attribute name="kind" type="string" use="required"/>
	    < assert test="if (isbn)
	        then publisher
	        else if (not(isbn))
	        then not(publisher)
	        else false" />
	</complexType>
	
	Both the 'isbn' and 'publisher' elements are optional, but we want to make sure that if one is entered then the other is too. The assertion above implements this rule.
	
	<u>NOTE 1:</u>
	The positioning of assertions is important.
		High.
		
		Assertions can provide a mechanism to declaratively express dependency rules in schemas rather than burying them in procedural code.

[1] Throughout this document, in an effort to simplify the examples, namespace prefixes have been omitted. E.g. <element> instead of <xsd:element>.

Feature	Description & Example	Relevance to Origo Schemas
	Assertions always look downwards. The root context of an assertion will be the type where the assertion is defined. So some thought is required in order to position a specific assertion correctly. There are two scenarios to be aware of when deciding where an assertion is placed: 1. Place the assertions on the complexType of the root element. This will give you maximum flexibility over what can be incorporated into your assertion. For example, if at a later date you may decide to incorporate additional factors into your assertion then since you've positioned it at the top of the XML tree you will be able to reference any data in the current document. 2. Place the assertions on the complexType that the assertion applied to. This makes the type and its assertions reusable across documents. More investigation is required to determine the recommended placement of assertions. **<u>NOTE 2</u>**: Assertions **cannot** make use of an XPath expression which makes reference to an external resource. For example an XPath expression like this is not permitted: `<assert test="publisher = doc('publishers.xml')//publisher" />`	

Feature	Description & Example	Relevance to Origo Schemas
	The doc() function in XPath is used to provide access to external documents. The statement above makes reference to a physically separate instance document (‘publishers.xml’) which provides a list of permitted publishers. This will not work in XSD 1.1. With assertions in XSD 1.1 the context is limited to the schema and the instance being validated.	

The <assertion> Facet	The <assertion> facet is used to constrain a simple type *e.g.*
	
	`<element name="evenPriceChangePercent" type="evenPriceChangePercentType"/>`
	`<simpleType name="evenPriceChangePercentType">`
	`  <restriction base="decimal">`
	`      <assertion test="$value <= 100 and $value >= 0"/>`
	`      <assertion test="$value mod 2 = 0"/>`
	`  </restriction>`
	`</simpleType>`
	
	The “evenPriceChangePercentType” simple type allows only even numbers between 0 and 100.
		High.
		
		Similar to the <assert> element used in complex types this can be used to specify additional validation rules.

Conditional Inclusion (Version Control)	XSD 1.1 introduces a new namespace, the version control namespace. By convention **vc:** is used to prefix items in this namespace.	Medium. This may be considered by Origo in the

Feature	Description & Example	Relevance to Origo Schemas
	vc:minVersion and **vc:maxVersion** may be placed as attributes on an element declaration to indicate which version of the **schema specification** the declaration was written to. **vc:typeAvailable** and **vc:typeUnavailable** may be placed as attributes on an element declaration to signal to a schema validator that a vendor-unique data type is being used by the element. **vc:facetAvailable** and **vc:facetUnavailable** may be placed as attributes on an element declaration to signal to a schema validator that a vendor-unique facet is being used by the element. *e.g.* There are two declarations of an element 'book'; 1) the first declaration is used by schema validators that implement version 1.1 (or later) of the schema specification; 2) the second declaration is used by schema validators that implement any version of the schema specification between 1.1 and 2.5: <element name="book" **vc:minVersion**="1.1"> *declare the Book element* </element> <element name="book: **vc:minVersion**="1.1" **vc:maxVersion**="2.5"> *declare the Book element* </element>	future as XSD 1.1 moves to XSD 1.2 or XSD 2.0 and onwards. At the moment, because XSD 1.0 does not support this, it would be of little use when used with XSD 1.1. Worth considering for the future when further releases of XSD evolve.

Feature	Description & Example	Relevance to Origo Schemas
	Schema pre-processing will remove the type definition which is not relevant. Note that this feature is provided to future proof XSD. It is of little use until a future version of XSD is created to operate alongside the new XSD 1.1 version. This feature has no connection with the individual schema versioning information recorded for each individual schema authored and published by Origo using the schema “version” attribute. Origo will continue with this practice to denote versioning of individual schemas.	
Substitution Group can Substitute with Multiple Elements	The substitutionGroup capability has been enhanced so that an element can substitute with multiple elements *e.g.* `<element name="metro" substitutionGroup="metrorail subway" type="xsd:NCName" />` The <metro> element is substitutable for either <metrorail> or <subway>.	Medium. Origo use the substitutionGroup approach for current schema developments that are implemented with the Origo Message Transmission Guidelines version 2 and above [10]. Introducing this feature into the Origo schemas would not provide significant benefit.
The yearMonthDuration data type	A yearMonthDuration value is a constrained version of the duration data type which only allows years and months to be specified *e.g.*	Medium. The duration data type is used sparingly by

Feature	Description & Example	Relevance to Origo Schemas
	<element name="event_duration" type="**yearMonthDuration**" /> --- <event_duration>**P1Y3M**</event_duration> <event_duration>**P15M**</event_duration>	Origo, but it is worth noting that the new yearMonthDuration data type is available in XSD 1.1.
The dayTimeDuration data type	A dayTimeDuration value is a constrained version of the duration data type which only allows a day and time to be specified *e.g.* <element name="conference_duration" type="**dayTimeDuration**" /> --- <conference_duration>**P35DT01H22M30S**</conference_duration> <conference_duration>**PT11H**</conference_duration>	Medium. The duration data type is used sparingly by Origo, but it is worth noting that the new dayTimeDuration data type is available in XSD 1.1.
New Facet – explicitTimezone	Use this with date data types to specify whether the time zone is required, the values can be 'required', 'prohibited' or 'optional'. *e.g.* <simpleType name='bare-date'> <restriction base='date'> <explicitTimezone value='prohibited'/> </restriction> </simpleType> <date>2013-05-04Z</date> would **fail** the schema validation check <date>2013-05-04+04:00</date> would **fail** the schema validation check<date>2013-05-04</date> would **pass** the schema validation check	Medium. This could be useful to ensure that the time-zone portion of dates is not entered where it is not expected.

Feature	Description & Example	Relevance to Origo Schemas

Target Namespace	Allows restriction of Complex Types from a foreign schema which has a different target namespace (targetNamespace).
	
	NOTE: This is only applies when using qualified elements and attributes (elementFormDefault=’qualified’ & attributeFormDefault=’qualified’) in the schema definition for the base type) *e.g.*
	
	Schema books.xsd:
	`<xsd:schema targetNamespace="http://www.books.org"`
	`      xmlns:xsd="http://www.w3.org/2001/XMLSchema"`
	`      xmlns:books="http://www.books.org"`
	`      `**`elementFormDefault="qualified"`**
	`     `**`attributeFormDefault="qualified">`**
	
	`    <xsd:element name="book" type="books:BookType"/>`
	`    <xsd:complexType name="BookType">`
	`       <xsd:sequence>`
	`          <xsd:element name="title" type="xsd:string"/>`
	`         <xsd:element name="author" type="xsd:string" maxOccurs="unbounded"/>`
	`         <xsd:element name="date" type="xsd:date"/>`
	`         <xsd:element name="isbn" type="xsd:string"/>`
	`         <xsd:element name="publisher" type="xsd:string"/>`
	`       </xsd:sequence>`
	`       <xsd:attribute name="attr1" type="xsd:string" use="required"/>`
	`       <xsd:attribute name="attr2" type="xsd:string" use="required"/>`
	`    </xsd:complexType>`
	`</xsd:schema>`
		Medium.
		
		Most of the Origo Standards use qualified elements and attributes (the only exception to this is the QNB Standards) thus allowing the use of this feature.
		
		This feature could be useful to those wishing to customise (via restriction) the Origo Standards (in particular the schema data patterns used in the Flexible Integration Toolkit [19]) and at the same time remain compliant with the Origo Standards.

Feature	Description & Example	Relevance to Origo Schemas

	Schema booksinlibrary.xsd
	`<xsd:schema targetNamespace="http://www.libraries.org"`
	`    xmlns:xsd="http://www.w3.org/2001/XMLSchema"`
	`    xmlns:lib="http://www.libraries.org"`
	`    xmlns:books="http://www.books.org"`
	`    elementFormDefault="qualified"`
	`    attributeFormDefault="qualified">`
	
	`    <xsd:import namespace="http://www.books.org" schemaLocation="books.xsd"/>`
	
	`   <xsd:element name=”book_in_library” type=”lib:BookInLibraryFromBooksSchema”/>`
	
	`    <xsd:complexType name="BookInLibraryFromBooksSchema">`
	`      <xsd:complexContent>`
	`        <xsd:restriction base="books:BookType">`
	`          <xsd:sequence>`
	`            <xsd:element name="title" type="xsd:string" targetNamespace="http://www.books.org" />`
	`            <xsd:element name="author" type="xsd:string" maxOccurs="2"`
	`                                    targetNamespace="http://www.books.org" />`
	`            <xsd:element name="date" type="xsd:date" targetNamespace="http://www.books.org" />`
	`            <xsd:element name="isbn" type="xsd:string" targetNamespace="http://www.books.org" />`
	`            <xsd:element name="publisher" type="xsd:string" targetNamespace="http://www.books.org"/>`
	`          </xsd:sequence>`
	`          <xsd:attribute name="attr1" type="xsd:string" use="required"`
	`                                    targetNamespace="http://www.books.org" />`
	`          <xsd:attribute name="attr2" type="xsd:string" use="required"`
	`                                    targetNamespace="http://www.books.org" />`
	`        </xsd:restriction>`
	`      </xsd:complexContent>`
	`   </xsd:complexType>`	

Feature	Description & Example	Relevance to Origo Schemas

	</xsd:schema>
	
	By adding the target namespace to the elements in the restricted type the original definition can be located and used accordingly.
	
	NOTE:
	In the example above the element called ‘book_in_library’ must use the namespace prefix **lib:** (‘http://www.libraries.org’), whereas all its child elements must use the namespace prefix **books:** (‘http://www.books.org’) *e.g.*
	
	<lib:book_in_library
	        xmlns:books="http://www.books.org"
	        xmlns:lib="http://www.libraries.org"
	        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
	        xsi:schemaLocation=http://www.libraries.org booksinlibrary.xsd
	        books:attr1="attr1" books:attr2="attr2">
	  <books:title>book title</books:title>
	  <books:author>author 1</books:author>
	  <books:author>author 2</books:author>
	  <books:date>2013-07-22</books:date>
	  <books:isbn>isbn</books:isbn>
	  <books:publisher>publisher</books:publisher>
	</lib:book_in_library>	

The <alternative>	Conditional Type Alternatives.	Low.

Feature	Description & Example	Relevance to Origo Schemas

Element	The <alternative> element is used to provide an element a choice of types, the actual type used in an instance document depends on the value of attributes, *e.g.*
	
	`<publication kind="`**book**`">`
	`    <title>Beginning XSLT</title>`
	`    <author>Jeni Tennison</author>`
	`    <date>2006</date>`
	`    <isbn>1-57851-777-X</isbn>`
	`    <publisher>Wrox</publisher>`
	`</publication>`
	
	`<publication kind="`**magazine**`">`
	`    <title>Computing</title>`
	`    <author>BCS</author>`
	`    <date>2013</date>`
	`    <genre>IT </genre>`
	`    <circulation>2500000</circulation>`
	`    <frequency>Monthly</frequency >`
	`</publication>`
	
	`<element name="publication" type="PublicationType">`
	`  <alternative test="@kind eq 'book'" type="BookType" />`
	`  <alternative test="@kind eq 'magazine'" type="MagazineType" />`
	`</element>`
	
	The content of <publication> depends on the kind of publication.
	If the value of the kind attribute is 'book' then <publication>'s type is BookType; if the value of the kind attribute is 'magazine' then its type is MagazineType; otherwise, its
		In the Origo Standards schemas content structures are described at the design stage with the provision of specific types as appropriate.
		
		On the occasions where this concept is required in the Origo schemas (the QNB schemas) the use of 'xsi:type' has been used in instance documents to denote the type of structure being presented.
		
		More recent Origo schema developments have avoided the use of "xsi:type" in favour of a prescriptive and predictable XML structure.
		
		Conditional type alternatives are, however, an improvement on 'xsi:type'.

Feature	Description & Example	Relevance to Origo Schemas
	type is PublicationType.	

Schema Wide Attributes (defaultAttributes)	defaultAttributes is used to specify a set of attributes that apply to every complexType in a schema document.
	
	`<schema`
	   `defaultAttributes="myDefaultAttributes"``>`
	`<element name="publication" type="PublicationType"/>`
	`<complexType name="PublicationType">`
	        `< sequence>`
	                `< element name="title" type="string" />`
	                `< element name="author" type="string" />`
	                `< element name="date" type="string" />`
	                `< element name="isbn" type="string" minOccurs="0"/>`
	                `< element name="publisher" type="string" minOccurs="0"/>`
	        `</ sequence>`
	        `< attribute name="kind" type="string" use="required"/>`
	`</complexType>`
	`<:attributeGroup name="myDefaultAttributes">`
	    `<attribute name="id" type="xsd:ID" use="required" />`
	  `</attributeGroup>`
	`</schema>`
	
	In the example above each complex type would also have a mandatory xsd:ID attribute associated with it.
	
	NOTE: Default attributes cannot be overridden. In the example above we cannot decide that some of the "id" attributes can be optional – they will all be mandatory based on the definition above.
		Low.
		
		In current schema development Origo use attributes very sparingly so this would not provide much benefit for those developments.
		
		However in older Origo Standards, for example the QNB schemas, we do have extensive use of attributes. Unfortunately their use is not consistent across a whole schema. The xsd:ID attributes, for example, are mostly optional but some are mandatory.

Feature	Description & Example	Relevance to Origo Schemas

Open Content	This allows open content to be specified throughout a defined structured *e.g.*
	
	<element name="publication" type="PublicationType"/>
	<complexType name="PublicationType">
	    <openContent mode="interleave">
	        <any processContents="skip"/>
	    </openContent>
	    < sequence>
	        < element name="title" type="string" />
	        < element name="author" type="string" />
	        < element name="date" type="string" />
	        < element name="isbn" type="string" minOccurs="0"/>
	        < element name="publisher" type="string" minOccurs="0"/>
	    </ sequence>
	    < attribute name="kind" type="string" use="required"/>
	</complexType>
	
	<publication kind="book">
	  <binding>hard cover</binding>
	  <title>Beginning Java<title>
	  <size>A4</size >
	  <author>Ivor Horton</author>
	  <date>2002</date>
	</publication>
	
	The <binding> and <size> elements have been included as 'open' content.
		Low.
		
		This does not provide the level of prescription necessary in most Origo schemas so is of little use.
		
		The <openContent> feature does provide additional facilities for implementers to customise schemas to suit their own purposes. However Origo have already documented a number of mechanisms which allow for extension of the Origo schemas – see the Origo Standards Implementation Guidelines [20]

Feature	Description & Example	Relevance to Origo Schemas

Vendor Unique Extensions	Software vendors can add their own data types and facets *e.g.*
	
	A vendor creates a new decimal data type and a facet for specifying the delimiter to be used in the decimal value:
	
	`<simpleType name="money">`
	`    <restriction base="`**`vendor:decimal`**`">`
	`        `**`<vendor:delimiter value="," />`**
	`    <restriction>`
	`</simpleType>`
	
	`<money>50000,00</money>`
	
	The vendor allows the comma to be used as the decimal delimiter.
		Low.
		
		This is not something Origo would encourage as the Origo Standards are used across a number of different platforms and technologies using different software tools.
		
		Origo also use standard data representations and only a restricted list of options when defining simple types e.g. min value, max value, max no of digits, max no of decimal places.

Inherited Attributes	Attributes can be declared to be inheritable and can then be used by descendant elements that contain <alternative> elements *e.g.*
	
	`<element name="publication" type="PublicationType"/>`
	`<complexType name="PublicationType">`
	`        <sequence>`
	`                <element name="title" type="string"/>`
	`                <choice>`
	`                        <element name="book" type="BookType"/>`
	`                        <element name="magazine" type="MagazineType"/>`
		Low.
		
		See Conditional Type Alternatives above. The same comments apply here.
		
		In the Origo Standards schemas content structures are described at the design stage with the provision of specific types as appropriate.

Feature	Description & Example	Relevance to Origo Schemas

	</choice>
	    </sequence>
	    <attribute name="lang" **inheritable="true"** type="Language" use="required" />
	</complexType>
	<complexType name="BookType">
	    <sequence>
	        <element name="isbn" type="string"/>
	        <element name="publisher" type="string">
	            <alternative test="@lang eq 'EN'" type="EnglishPublisher" />
	            <alternative test="@lang eq 'FR'" type="FrenchPublisher" />
	        </element>
	    </sequence>
	</complexType>
	<complexType name="MagazineType">
	    <sequence>
	        <element name="circulation" type="int"/>
	        <element name="frequency" type="string"/>
	    </sequence>
	</complexType>
	<simpleType name="Language">
	    <restriction base="string">
	        <enumeration value="EN"/>
	        <enumeration value="FR"/>
	    </restriction>
	</simpleType>
	<xsd:simpleType name="EnglishPublisher">
	    <restriction base="xsd:string">
	        <enumeration value="Wrox"/>
	            <xsd:enumeration value="McMillan"/>
	        </xsd:restriction>
	</xsd:simpleType>	

Feature	Description & Example	Relevance to Origo Schemas

	<xsd:simpleType name="FrenchPublisher">
	        <xsd:restriction base="xsd:string">
	                <xsd:enumeration value="Bayard"/>
	                <xsd:enumeration value="Le Castor Astral"/>
	        </xsd:restriction>
	</xsd:simpleType>
	
	The XML below would throw a validation error because the <publisher> element contains a value of type FrenchPublisher not EnglishPublisher.
	
	<publication lang="**EN**">
	  <title>Beginning XSLT<title>
	  <book>
	    <isbn></isbn>
	    <publisher>Bayard</publisher>
	  </book>
	</publication>	

Unordered Content using the <all> Element	The <all> element has been enhanced to allow elements with multiple occurrences. Also, <all> can have the wildcard, <any>, at any child position *e.g.*
	
	<element name="book" type="BookType/>
	<complexType name="BookType">
	    <**all**>
	      <any maxOccurs="unbounded" processContents="skip"/>
	      <element name="author" type="xsd:string" **maxOccurs="unbounded"**/>
	      <element name="title" type="xsd:string"/>
	      <element name="date" type="xsd:string"/>
		Low.
		
		Origo no longer use <all> or <any> in order to be as prescriptive as possible. The only exception to this is the existing QNB Standard which allows the use of <tpsdata> elements which use <any> content.
		
		The enhancements to <all> and <any>

Feature	Description & Example	Relevance to Origo Schemas

	`<element name="isbn" type="xsd:string"/>`
	`<element name="publisher" type="xsd:string"/>`
	`</`**`all`**`>`
	`</complexType>`
	
	The content of BookType is: any number of extension elements, any number of authors, title, date, isbn, and publisher, and they can be arranged in any order in instance documents.
	
	`<book>`
	`<date>date0</date>`
	`<isbn>isbn0</isbn>`
	`<publisher>publisher0</publisher>`
	`<author>author</author>`
	`<title>title0</title>`
	`<price>25.00</price>`
	`</book>`
	
	The <price> element has been added via the <any> within the <all>. The elements can appear in any order with the use of <all>.
		provide additional facilities for implementers to customise schemas to suit their own purposes. However Origo have already documented a number of mechanisms which allow for extension of the Origo schemas – see the Origo Standards Implementation Guidelines [20].

New Attributes of the <any> and <anyAttribute> Wildcard Elements	The <any> and <anyAttribute> wildcard elements have been enhanced with additional attributes that allow you to indicate the kind of extension elements or attributes ***not*** allowed. The notNamespace attribute is used to indicate the namespace that extension elements or attributes cannot come from. The notQName attribute is used indicate an element or attribute that is not allowed *e.g.* `<any `**`notNamespace`**`="http://www.example.org"/>`	Low. Origo no longer use <all> or <any> in order to be as prescriptive as possible. The only exception to this is the existing QNB Standard which allows the use of <tpsdata>

Feature	Description & Example	Relevance to Origo Schemas

	`<anyAttribute **notNamespace**="http://www.example.org"/>`
	
	`<any **notQName**="xsl:value-of"/>`
	
	The first wildcard does not allow extension elements from the http://www.example.org namespace.
	
	The second wildcard does not allow extension attributes from the http://www.example.org namespace.
	
	The third wildcard does not allow xsl:value-of as an extension element.
		elements which use <any> content.
		
		The enhancements to <all> and <any> do however provide improved facilities for those customising the Origo schemas for their own purposes.

More Flexible Rules for Wildcards	Wildcards (<any>) are important tools for extensible languages, but in XSD 1.0, it is difficult or impossible to use wildcards near optional content. XSD 1.1 is much more flexible *e.g.*
	
	`<element name="book" type="BookType"/>`
	`<complexType name="BookType">`
	`    <sequence>`
	`        <element name="title" type="string"/>`
	`        `**`<element name="numPages" type="int" minOccurs="0"/>`**
	`        `**`<any minOccurs="0" maxOccurs="unbounded"/>`**
	`    </sequence>`
	`</complexType>`
	
	`<book>`
		Low.
		
		Origo no longer use <all> or <any> in order to be as prescriptive as possible. The only exception to this is the existing QNB Standard which allows the use of <tpsdata> elements which use <any> content.
		
		The enhancements to <all> and <any> do however provide improved facilities for those customising the Origo schemas for their own purposes.

Feature	Description & Example	Relevance to Origo Schemas
	<title>The Origin of Wealth</title> <numPages>321</numPages> <reviews>Excellent</reviews> <binding>Hardcover</binding> </Book> In XSD 1.0 the element declaration shown here is not legal, because there are documents with elements like <numPages> that could match either the explicit element declaration or the wildcard. In XSD 1.1, this schema is valid, and the <numPages> element is validated as an integer by the element declaration. The <reviews> and <binding> elements are validated against the <any> wildcard.	
Enhanced Usage of the ID data type	In XML Schema 1.1 an element can have multiple attributes of type ID and the ID type can have a fixed or default value *e.g.* <element name="stereo" type="StereoType"> <complexType name="StereoType"> <sequence> … </sequence> **<attribute name="model-number" type="ID" use="required" />** **<attribute name="serial-number" type="ID" use="required" />** </complexType> <attribute name="food" type="**ID**" **fixed="Popcorn"** />	Low. Origo has an internal standard where by all ID attributes are given the name 'id'. There is no immediate requirement to allow an element to have more than one id. Current schema design guidelines at Origo prohibit the use of the 'fixed' attribute on element specifications. The same position should probably be taken regarding the use

Feature	Description & Example	Relevance to Origo Schemas
		of fixed values for attributes.

The <override> element	The <override> element replaces the XSD 1.0 <redefine> element, which has been deprecated. The <override> element is used to replace the contents of a globally declared item in another schema *e.g.*
	
	Office-calendar.xsd declares a <meeting> element with content <start-time>, <end-time>, and <room-number>. Conference-calendar.xsd overrides <meeting>'s content with <track-id>, <speaker>, and <room-capacity>:
	
	`<element name="meeting">`
	    `<complexType>`
	        `<sequence>`
	           `<element name="start-time" type="time" />`
	           `<element name="end-time" type="time" />`
	           `<element name="room-number" type="string" />`
	        `</sequence>`
	    `</complexType>`
	`</element>`
	
	Conference-calendar.xsd overrides the meeting element:
	
	<**override** schemaLocation="office-calendar.xsd">
	    `<element name="meeting">`
	        `<complexType>`
	           `<sequence>`
	              `<element name="track-id" type="string" />`
	              `<element name="speaker" type="string" />`
	              `<element name="room-capacity" type="int" />`
		Low.
		
		Origo do not permit the use of <redefine> in the Origo Standards so are unlikely to use <override>.
		
		XSD will support <redefine> until the next major release of XSD (e.g. 2.0) when the deprecated <redefine> element will be removed.

Feature	Description & Example	Relevance to Origo Schemas

	</sequence>
	         </complexType>
	    </element>
	</**override**>	

The anyAtomicType data type	The anyAtomicType is the union of the value spaces of all the primitive types *e.g.* <element name="value" type="**anyAtomicType**" /> --- <value xsi:type="string">**Hello World**</value> <value xsi:type="decimal">**12.36**</value> <value xsi:type="boolean">**true**</value>	Low. This is similar to anySimpleType but more restrictive. Origo do not use anySimpleType, preferring to be prescriptive, so are unlikely to use anyAtomicType.
The dateTimeStamp data type	A dateTimeStamp value is identical to the dateTime data type, except it requires time zone be specified *e.g.* <element name="date_of_birth" type="**dateTimeStamp**" /> --- <date_of_birth>1976-06-21T16:04:00-**6:00**</date_of_birth> <date_of_birth>1980-01-01T24:00:00-**6:00**</date_of_birth>	Low. The time zone is optional in dateTime but in practise it is very rarely used. Origo would be unlikely to make use of this new data type.
The error data type	The xsd:error data type is used to trigger a schema validation error. It may be used wherever a type is used *e.g.* <element name="publication" type="PublicationType"> <alternative test="@kind eq 'book'" type="BookType" />	Low. As stated above it is unlikely that Origo would use the alternative element, and as

Feature	Description & Example	Relevance to Origo Schemas

	`<alternative test="@kind eq 'magazine'" type="MagazineType" />`
	`<alternative test="(@kind ne 'book') and (@kind ne 'magazine')" type="xsd:error" />`
	`</element>`
	
	If the 'kind' attribute contains anything other than 'book' or 'magazine' then a schema validation error will be triggered.
		the error type is associated with the alternative element it follows that similarly it is unlikely that Origo could make use of the error type.

12 Appendix B – Documentation References

[1] XML
http://en.wikipedia.org/wiki/XML

[2] XML Schema
http://en.wikipedia.org/wiki/Xsd

[3] XPath 2.0
http://en.wikipedia.org/wiki/XPath_2.0

[4] Schematron
http://en.wikipedia.org/wiki/Schematron

[5] XML Data Binding
http://en.wikipedia.org/wiki/XML_Data_Binding

[6] W3C XML Schema 1.0 Specification
http://www.w3.org/TR/xmlschema-1/

[7] XML Schema Primer
http://www.w3.org/TR/xmlschema-0/

[8] W3C XML Schema 1.1 Specification
http://www.w3.org/TR/xmlschema11-1/

[9] XForms
http://en.wikipedia.org/wiki/XForms

[10] Origo Message Transmission Guidelines
http://www.origoservices.com/O_StandardDetail.aspx?Standard=HTTP%20Message%20Transmission&Version=2.1&Cat=Life%20and%20Pensions

[11] Saxon Support for XML Schema 1.1
http://saxonica.com/documentation9.4-demo/html/schema-processing/schema11/

[12] XERCES Support for XML Schema 1.1
http://xerces.apache.org/xerces2-j/xml-schema.html#supported-schema-1.1-features

[13] JAXP
http://docs.oracle.com/javase/tutorial/jaxp/

[14] Oxygen XML Editor
http://www.oxygenxml.com/

[15] Microsoft .NET Framework
https://en.wikipedia.org/wiki/.NET_Framework

[16] Liquid XML
http://www.liquid-technologies.com/whatsnew.aspx

[17] Altova XmlSpy
http://www.altova.com/xmlspy.html

[18] Origo Receive Automatic Enrolment Employee List Standard
http://www.origoservices.com/O_StandardDetail.aspx?Standard=Receive%20Automatic%20Enrolment%20Employee%20List&Version=1.0%20Draft%20Final&Cat=Life%20and%20Pensions

[19] Origo Flexible Integration Toolkit (Pre-population)
http://www.origoservices.com/fit

[20] Origo Standards Implementation Guidelines
http://www.origoservices.com/osig