

Why are simple code lists so complex?

Anthony B. Coates
Senior Partner, Miley Watts LLP
abcoates@mileywatts.com

Presented to the New York XML Special Interest Group
9th May 2007

Contents

- ↻ Introduction
- ↻ What a difference a day makes
- ↻ Turning the tables on code lists
- ↻ The code list is in the eye of the beholder
- ↻ Keeping it real
- ↻ Genericode Structure
- ↻ Example genericode 1.0 document
- ↻ Next steps – managing change
- ↻ Conclusion

Introduction

- ⇒ The wonderful thing about XML is that it lets you choose your own vocabulary
- ⇒ Arbitrary choice of element and attribute names
- ⇒ However, arbitrary choice is **too much** for most applications – it simply costs too much to code defensively for arbitrary content
- ⇒ That means we need to restrict ourselves to specific vocabularies in our element and attribute naming

Introduction

- ↻ Now we face the problem of aligning and managing those vocabularies to achieve **interoperability** (*Walter Perry excepted*)
- ↻ However, it is not just element & attribute names that need to be semantically unambiguous & aligned for interoperability
- ↻ Content models for elements need to be aligned

Introduction

- ↻ The lexical form of element and attribute text content also needs to be aligned, i.e. **simple data items need to be represented the same way**
- ↻ This latter point is **more important for applications** than is alignment of content models or element & attribute names

Introduction

- ↻ The idea that alignment of the lexical form of element and attribute text content is **more important** than alignment of content models or element & attribute names isn't obvious

Introduction

- ⇒ However, when you talk to the people who build and maintain back-end systems (e.g. in finance, where my background is), it becomes clear that changing code that handles low-level datatypes (often closely tied to a database) is typically **more expensive** than changing code to manage a change in a content model or to element or attribute names
- ⇒ So simple datatype formats **have to be aligned**

Introduction

- ↻ The wide popularity of the W3C XML Schema datatypes has helped somewhat, in terms of aligning date/time and numeric formats
- ↻ This presentation focusses on another key simple datatype – *code lists, aka enumerations, aka controlled vocabularies*
- ↻ For data-oriented XML particularly, code lists are as important as element & attribute names – they form part of the *complete vocabulary* of the document
- ↻ However ...

Introduction

- ↻ It is said that it takes only a minute to decide to spend a million dollars, because **nobody** really **understands** what a million dollars is
- ↻ However, people can argue about the choice of office paper clips for hours, because **everybody** is an **expert** on paper clips
- ↻ Code lists are similar, because they are so **obviously simple**, and everyone knows everything about them

Introduction

- ↻ If code lists were really so simple and obvious, there would be a **single, well-known and accepted** way of handling them in XML
- ↻ There is no such agreed solution, though
- ↻ The problem is that while code lists are a well understood concept, people **don't actually agree** on exactly what code lists are, and how they should be used

What a difference a day makes

- ⇒ What is a code list, then?
- ⇒ Most people would agree that the following is a code list:
{ 'SUN', 'MON', 'TUE', 'WED', 'THU', 'FRI', 'SAT' }
- ⇒ This is a perfectly reasonable set of alphabetic codes for representing days of the week
- ⇒ However, so is:
{ 'Sun', 'Mon', 'Tue', 'Wed', 'Thu', 'Fri', 'Sat' }

What a difference a day makes

- ⇒ These two code lists are very similar, but certainly not identical
- ⇒ That said, they can both be used to represent the days of the week
- ⇒ Of course, you could also use:
`{ 'Dim', 'Lun', 'Mar', 'Mer', 'Jeu', 'Ven', 'Sam' }`

What a difference a day makes

⇒ Then again, you could use:

{0, 1, 2, 3, 4, 5, 6}

which is suitable as a computer representation, e.g. for a database column

⇒ On the other hand:

{'S', 'M', 'T', 'W', 'T', 'F', 'S'}

is not suitable as a code list for the days of the week, because the values are not unique

What a difference a day makes

- ⇒ Now suppose that you are using codes to represent days of the week in an application
- ⇒ You are displaying the days of the week using 3-letter abbreviations in English or French
- ⇒ Should `{ 'Sun', ... }` and `{ 'Dim', ... }` be considered to be code lists?

What a difference a day makes

- ⇒ Should they be considered to be display values that would be keyed to either { 'SUN', ... } or { 0, 1, ... }?
- ⇒ Think about it!

What a difference a day makes

- ↻ The fact is, they could be either code lists or display values
- ↻ A value which is a code in one context might only be an associated value for a code in another context
- ↻ Nothing privileges any of these code lists over the others in terms of ability or suitability
- ↻ Well, excepting { 'S', 'M', ... } which is not a suitable list

What a difference a day makes

- ⇒ There is a **choice** of code lists that can be used
- ⇒ The answer to the question "which choice is the best?" depends on the needs of each **particular situation**
- ⇒ No wonder it is so hard to agree on "what is a code list"

What a difference a day makes

- ⇒ Even agreeing on “**what are the codes**” is non-trivial, because there is no globally unique, unambiguous, one-size-fits-all answer
- ⇒ There are choices to be made, and any broad, general solution has to offer the necessary **freedom of choice**

Turning the tables on code lists

- ↻ What the “days of the week” example showed was that for each **distinct entry** in a list, there are many possible **associated values**
- ↻ Some of those associated values are suitable for use as unique codes, some are not
- ↻ This leads to a tabular model, where each row of the table represents a distinct entry, and each column represents an associated (metadata) value:

Turning the tables on code lists

numeric (key)	english, uppercase (key)	english, mixed case (key)	french, mixed case (key)	english, single character
0	SUN	Sun	Dim	S
1	MON	Mon	Lun	M
2	TUE	Tue	Mar	T
3	WED	Wed	Mer	W
4	THU	Thu	Jeu	T
5	FRI	Fri	Ven	F
6	SAT	Sat	Sam	S

Turning the tables on code lists

- ⇒ Notice that the first 4 of the 5 columns are labelled as **key** columns
- ⇒ This means that the values in those columns can be used to uniquely identify the rows, and hence they can be used as code list values

Turning the tables on code lists

- ⇒ The term key is used here in the same fashion as for a **relational database table**
- ⇒ This is the most common case, where a single column can be used as a key. However, consider the following modification:

Turning the tables on code lists

numeric (key)	english, uppercase (key)	english, single character #1	english, single character #2
0	SUN	S	U
1	MON	M	O
2	TUE	T	U
3	WED	W	E
4	THU	T	H
5	FRI	F	R
6	SAT	S	A

Turning the tables on code lists

- ⇒ Here, the first two columns are each a key column
- ⇒ The last two columns are not individually key columns, but together they form a **compound key**
- ⇒ While the two individual columns do not contain unique values, the **pair of values** is unique within each row

Turning the tables on code lists

- ⇒ This is again similar to what happens in some relational databases, that a key for the rows need not be constructed from a single column, but instead may be constructed by **combining** two or more columns
- ⇒ Finally, there is no reason why a column should only contain simple values like strings or numbers
- ⇒ A column could also contain a complex compound group of data, such as a fragment of XML:

Turning the tables on code lists

numeric (key)	english, uppercase (key)	XHTML
0	SUN	Sunday
1	MON	<i>Monday</i>
2	TUE	Tuesday
3	WED	<i>Wednesday</i>
4	THU	Thursday
5	FRI	<i>Friday</i>
6	SAT	Saturday

Turning the tables on code lists

- ⇒ Notice that the final XHTML column is not marked as a key column
- ⇒ The values are unique, so it certainly could be used as a key column
- ⇒ However, sometimes you may not wish to mark a column as a key column, even if the values are unique

Turning the tables on code lists

- ⇒ The values in the column may not make particularly suitable keys
- ⇒ They might be too long to process quickly and conveniently
- ⇒ They might not be able to be used in a particular context, such as for an XML attribute value

Turning the tables on code lists

- ⇒ Also, it may be that while the values in a particular column are unique now, there is no guarantee or expectation that they will **remain unique** as the code list grows or changes in future

The code list is in the eye of the beholder

- ⇒ Once you see the tabular nature that underlies the information that can be associated with code lists, it becomes clear why they can be a source of so much debate
- ⇒ Different users need **different subsets** of the code list information
- ⇒ People are inclined to **assume** that the information they need is all the information that anyone needs

The code list is in the eye of the beholder

- ↻ That kind of thinking doesn't work with code lists, because code lists are sufficiently **generic** a concept that they are used across messages/documents, applications, and databases
- ↻ The code list details that you need for the XML schemata often will **not be exactly the same** as the details that you need for your database or your application

The code list is in the eye of the beholder

- ⇒ If the code list information cannot be shared easily across these different areas of the business, the result is:
 - ⇒ duplication of effort
 - ⇒ potential **loss of synchronisation** between different implementations of the same code list

The code list is in the eye of the beholder

- ↻ The **XML schema** may only require a set of 3-letter codes to represent the code list
- ↻ The **database** may require a set of numeric codes, plus display labels (possibly in different languages)
- ↻ The **application** may need to know which 3-letter code corresponds to which numeric code, so that it can process the XML and update the database

The code list is in the eye of the beholder

- ⇒ All of this code list information needs to be able to be stored together in a **single representation** of the code list, so that all usages of the code list can be generated from the same source information

The code list is in the eye of the beholder

- ⇒ One last piece of experience from databases is that support for **undefined values** will be required
- ⇒ Sometimes users will have values that need to be associated with some of the codes in a code list, but won't have a value to associate with each and every code
- ⇒ In that case, the concept of a undefined value is needed

Keeping it real

- ↻ The initial impetus for looking at how to represent code lists came from two financial XML specifications, **FpML** (Financial Products Markup Language) and **MDDL** (Market Data Definition Language)
- ↻ FpML defined the approach of keeping enumerations out of the core Schema by using **schemes**

Keeping it real

- ↻ The idea is that the code list from which an element value is taken is indicated via a **scheme** attribute containing a URI which represents the scheme (code list)
- ↻ Same as the way that URIs are used to represent XML namespaces
- ↻ This was done so that a new version of FpML did not have to be released just because an **externally controlled** enumeration had changed (e.g. a currency or country code)

Keeping it real

- ⇒ Also straightforward for groups of users to use alternate code lists as appropriate
- ⇒ `<Currency scheme="http://www.fpml.org/ext/iso4217-2001-08-15">USD</Currency>`
- ⇒ One thing that FpML never formally defined was what happens if you **dereference** a scheme URI (i.e. type it into a Web browser)

Keeping it real

- ↻ MDDL copied the FpML scheme approach
- ↻ Like FpML, no formally defined scheme format
- ↻ Has a **different** draft XML format for schemes
- ↻ A key aim of the generic code list model was to produce a format that could be shared by FpML and MDDL

Keeping it real

- ⇒ Note that no XML schema language currently supports the notion of schemes in this sense, so applications have to implement their **own validation** of codes against schemes at present
- ⇒ How many of you have the same issue – you want to validate codes in an XML document against a defined list, but you don't want that list to be **embedded** in the Schema you use?
- ⇒ How many of you would like a **standard software layer** that handles this?

Keeping it real

- ↻ Around the same time, **UBL** (Universal Business Language), as working on its own approach to code lists, so it again seemed like a good idea to try and unify all of these efforts, and reduce the duplication
- ↻ So, I have been working with UBL, FpML, and MDDL to see what common ground exists for a common code list format

Keeping it real

- With the support of Jon Bosak of Sun, who chairs UBL, the genericcode work has now been moved to its own OASIS Technical Committee, the Code List Representation TC
- http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=codelist
- The task of the TC is to define an XML interchange format for code lists and related information

Genericode Structure

- ⇒ OASIS genericode will have a normative W3C XML Schema
- ⇒ The Schema diagrams are too detailed for use in a presentation, so I will summarise the structure in a series of slides

Genericcode Structure

- ⇒ There are 3 kinds of documents that the Schema supports, and 3 global elements that can be the document root
- ⇒ **<ColumnSet>** — defines a set of columns and keys that can be re-used in code list definitions
- ⇒ **<CodeList>** — definition of a simple or derived code list

Genericcode Structure

- ⇒ **<CodeListSet>** — a set of code list versions, used to define a code list **configuration**
- ⇒ Think back to the earlier tabular examples:

Genericode Structure

numeric (key)	english, uppercase (key)	english, single character #1	english, single character #2
0	SUN	S	U
1	MON	M	O
2	TUE	T	U
3	WED	W	E
4	THU	T	H
5	FRI	F	R
6	SAT	S	A

Genericode Structure

- ↻ A **column** is a class of value (metadata) that can be associated with a code in a list, e.g. the code itself, or a human readable name
- ↻ A column has an identifier, and can have human readable and machine processable metadata (e.g. names)
- ↻ The W3C XML Schema **annotation** structure is copied for this

Genericode Structure

- ↻ Each column has a data type. By default, the W3C XML Schema datatypes are assumed, but the RELAX NG **datatype library** mechanism is copied so that alternative sets of datatypes could be used
- ↻ A **key** is a set of one or more columns whose value(s) can be used to **uniquely identify** an item in a code list

Genericode Structure

- ⇒ Any code list must have at least one key (and hence at least one column), but there is **no upper limit** on the number of keys that can be defined
- ⇒ Also, there is no concept of a **preferred key**. Choice of a key is assumed to be a **late binding** between a code list and a particular contextual usage
- ⇒ Like columns, each key has a unique identifier and optional metadata

Genericcode Structure

- ↻ Columns and keys are defined **either** in a column set or code list definition. They can be re-used from either in any code list definition
- ↻ A simple code list is a tabular definition of the contents of a code list
- ↻ Each item in the code list is represented by a **<Row>** element, and each column is represented by a **<Value>** in that row

Genericode Structure

- ⇒ A value can be associated with an explicit column; if not, the column is **inferred** from the order of both the values and columns in the table
- ⇒ A value must be provided for each **required** column, but need not be for an **optional** column, i.e. optional columns can contain undefined (null/nil) values
- ⇒ Only **required columns** can be used for keys

Genericode Structure

- ⇒ All genericode documents have the following common features:
 - ⇒ user-defined human and/or machine readable metadata can be added throughout the documents (validatable)
 - ⇒ **code lists** (etc.) and their **specific versions** are identified by separate URIs
 - ⇒ URIs can be provided as possible locations from which to download a definition
 - ⇒ dereferencing of identifier URIs is **discouraged**

Example genericode 1.0 document

```
⦿ <?xml version="1.0" encoding="UTF-8"?>
<gc:CodeList
  xmlns:gc="
    http://docs.oasis-open.org/codelist/ns/genericode/1.0/
  ">
<Identification>
  <ShortName>
    CountryIdentificationCode
  </ShortName>
  <Version>1.0</Version>
```

Example code list document

```
⊕ <CanonicalUri>  
  urn:oasis:names:specification:ubl  
:schema:xsd:CountryIdentificationCode  
</CanonicalUri>  
<CanonicalVersionUri>  
  urn:oasis:names:specification  
  :ubl:schema:xsd  
  :CountryIdentificationCode-1.0  
</CanonicalVersionUri>  
</Identification>
```

Example code list document

```
<ColumnSet>  
  <Column  
Id="CountryIdentificationCodeContent"  
  Use="required">  
  <ShortName>  
    CountryIdentificationCodeContent  
  </ShortName>  
  <Data Type="token" />  
</Column>  
<Column Id="CodeName" Use="required">  
  <ShortName>CodeName</ShortName>  
  <Data Type="string" />  
</Column>
```

Example code list document

```
⊕ <Key Id="
  CountryIdentificationCodeContentKey
">
  <ShortName>
    CountryIdentificationCodeContentKey
  </ShortName>
  <ColumnRef Ref="
    CountryIdentificationCodeContent
  "/>
</Key>
</ColumnSet>
```

Example code list document

```
⊕ <SimpleCodeList>
  <Row>
    <Value ColumnRef="
      CountryIdentificationCodeContent
    ">
      <SimpleValue>AD</SimpleValue>
    </Value>
    <Value ColumnRef="CodeName">
      <SimpleValue>ANDORRA</SimpleValue>
    </Value>
  </Row>
```

Example code list document

```
➔ <Row>
  <Value ColumnRef="
    CountryIdentificationCodeContent
  ">
    <SimpleValue>ZW</SimpleValue>
  </Value>
  <Value ColumnRef="CodeName">
    <SimpleValue>ZIMBABWE</SimpleValue>
  </Value>
</Row>
</SimpleCodeList>
</gc:CodeList>
```

Example code list document

- ⇒ That example showed you most of what you need to know to use genericode
- ⇒ There isn't much you need to know in order to get started
- ⇒ As much as was feasible, it was designed so that the simplest examples of code lists are simple to represent in XML

The only constant is change

- ↻ Code lists change
- ↻ For a code list representation to be most useful, it has to account for the fact that the code lists will **change over time**
- ↻ The code list representation has to support changes between versions of a code list
- ↻ Not all changes to a code list are version changes, however

The only constant is change

- ↻ Some changes may be local changes to a distributed code list
- ↻ The ISO 3-letter currency code list contains **GBP** for British Pounds. However, prices on the London Stock Exchange are normally quoted in pence
- ↻ This has led to the practice of adding an extra code to the standard ISO list (e.g. **GBp**, **GBX**) in order support pence as well as pounds

The only constant is change

- ⇒ This kind of customisation is far from uncommon
- ⇒ The utility of any code list representation is greatly reduced if it does not cater for **local modifications** of code lists
- ⇒ In terms of the **tabular model** of code lists that has been shown, the following are typical local customisations:

The only constant is change

- ⇒ Add or remove a **row** (the set of values associated with a distinct entry);
- ⇒ Add or remove a **column** (a type of value associated with each distinct entry);
- ⇒ Add or remove a **key**. Adding a key involves specifying the column(s) to be used for the key;

The only constant is change

- ⇒ Add **one or more rows** from a set of code lists together. This implies that the columns are the same, or that null values will be used as required;
- ⇒ Add **one or more columns** from a set of code lists together. This implies that there is at least one key in common in each row, and that there are no keys with conflicting values;

The only constant is change

- ⇒ Remove **rows** for which a key value matches the key value in another code list (this can be used to delete a particular **pre-defined subset** of the codes);
- ⇒ Modify a cell **value** (one of the values associated with a code). Note that this may impact whether the affected column can be used as a key or not;
- ⇒ Create a **derived column** whose values are generated from the values in other columns

The only constant is change

- ↻ There should also be a way for users to understand easily **how** their local code list was **derived** from one or more other code lists
- ↻ There needs to be an **audit trail**
- ↻ This means that the code list representation needs to model the ways that code lists are **modified**
- ↻ It also needs to be clear how to **regenerate** a derived code list when the underlying code lists change

The only constant is change

- ↻ Genericode 1.0 does **not** support derived code lists
- ↻ This was to keep the spec shorter and get a usable approved 1.0 version out sooner (for UBL and FpML particularly)
- ↻ The current plan is that support for derived code lists (which was part of the 0.4 draft) will be **re-introduced** into version 2.0
- ↻ What follows is the version 0.4 **draft approach**

The only constant is change

- A **derived** code list is represented as a sequence of code list operations that can be audited and repeated
- Operation types: **<ColumnSetExclusion>**,
<ColumnSetInclusion>,
<ColumnSetMatch> (required inclusion),
<ColumnSetUnion>, **<RowExclusion>**,
<RowInclusion>, **<RowMatch>**, **<RowUnion>**

The only constant is change

- ◉ A single derived code list definition can contain an **arbitrary depth** of operations, including embedded code list definitions

Genericode Status

- The genericode 1.0 Committee Draft will shortly be released for public review
- Watch OASIS for announcements (and/or leave a business card with G. Ken Holman or me)
- Your comments and questions are very welcome

Conclusion

- ↻ In this presentation, we have examined the reasons why code list models are often **too narrow**
- ↻ A **tabular model** of a code list, designed to cover a broad range and requirements and usages, is the basis of the OASIS Code List Representation TC's genericcode XML format

Conclusion

- ↻ Genericode captures the fact that code lists may be represented by **different sets of codes** in different circumstances, and that the choice of what is the code and what is associated content can **change** from one usage to another
- ↻ Future versions of genericode are planned to be capable of representing the steps required to **derive a code list** from a set of existing code lists

Conclusion

- ⇒ OASIS Code List Representation TC:
http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=codelist
- ⇒ From the TC page, you can view the document archive
- ⇒ You can also view the comments archive, or add a new comment
- ⇒ Original genericcode proposal (2004) with UML diagrams:
<http://www.idealliance.org/proceedings/xml04/abstracts/paper86.html>
- ⇒ Also <http://www.genericcode.org/>