SAP®
Intern

Elements versus Attributes

20. March 2002
Gunther Stuhec
Verteiler: UBL-Group

1   Introduction

A common cause of confusion, or at least uncertainty, in the design of a schemas is the choice between specifying parts of the document as elements or attributes. Elements and Attributes are both containers for information. Many times the choice between an Element and an Attribute seems very arbitrary, almost matter of style.

There is some information that could go either way. For example, *Country* could be an Attribute or an Element. Neither way is right or wrong, it is just a choice. While the choice may indeed be arbitrary in some cases, the 'typical' roles of Elements and Attributes and the different types of content models and constraints of these two containers will be explained in this document very shortly.

2   Characteristics

The fundamental difference between Elements and Attributes in XML 1.0 is to be define the limits of what the two containers can be used for. It means that elements can contain child elements as well as content and attributes can only hold content only**.** The distinction between attribute and content element then becomes the distinction between an attribute and a containment relationship with another object.

The following table shows the elementary differences of Elements and Attributes:

Elements	Attributes
Can have child Elements nested within them	Can't have nested Elements or Attributes; can contain only strings, or lists of strings
Typically used for structured data items but can be and are used for simple data items as well	Typically used for "atomic" items of data
Elements must appear in the order specified in the schema, but may appear several times.	Each Attribute of a particular Element can only be specified once, but more than one Attribute inside of one Element can be specified in any order
Elements usually represent the natural, core content, which would generally appear in every	Attributes represent data of secondary importance; often metadata?

printout/display?	
(Sub-)Elements usually represent parts of an Element	Attributes usually represent properties of an Element

2.1 Elements

Elements are logical units of information in a schema. They represent information objects. Elements either contain information (text), or have a structure of subelements. Therefore elements are good for representing structurally significant information.

Elements are more extensible than attributes in an evolving standard because elements can contain other elements or substructures directly while attributes cannot. If a concept is defined as an attribute initially, and then needs to be expanded to hold fine-grained information, it must be changed to an element to be modeled correctly.

Elements can have attributes attached to them as metadata, while attributes cannot. Elements are repeatable within the same container structure, but attributes can only appear once in the attribute list of an element. In addition, if order of occurrence is significant, elements are the only option because attributes do not have order.

2.2 Attributes

Attributes are atomic, referentially transparant characteristics of an object that have no identity of their own. Generally this corresponds to primitive data types (e.g., Strings, Date, etc.). Taking a more logical view, an attribute names some characteristic of an object that models part of its internal state, and is not considered an object in its own right. That is, no other objects have relationships to an attribute of an object, but rather to the object itself.

Attributes can be divided into the following types:
- The type of attribute that relating to element identification (ID and IDREF type attributes, and those attributes of type CDATA that have application-specific identification rules, such as the name attribute of the A element in HTML)
- Those containing tokens that identify one or more contexts in which the element applies, or which identify one or more options to be used during processing of the element (entity names, notation names, name tokens or values from a predefined set of tokens)
- Those tokens that carry data to be used as part of the application (typically CDATA type attributes).
- Attributes can also be describing the characteristics of information: a property of an information object. For example, notation attributes clearly define the coding of the data within the element, and so clearly control the processing of the contents. Similarly entity attributes clearly identify external, unparsed, entities that will need to be processed according to the rules applicable to the notation defined in the entity declaration.

The general characteristics of attributes are:
- Attribute values can have no substructure

- Attributes are unordered, so there is no standard way to specify that one attribute's value should precede the other's (there is no guarantee that an API will give you the attributes in the same order that you specified them)
- Attributes can only contain multiple values if they are tokens (*e.g.*, NMTOKEN) or references to other elements (*e.g.*, IDREFS)
- Attributes can only describe structures by using for example "xsi:type" and they can link to them (IDREF or ENTITY) but they cannot contain subelements directly in markup.

3   Advantages and Disadvantages of Attributes

It is much more easier to describe any general rule for using attributes esspecially, if the advantages and disadvantages are putted into the opposite before.

The advantages of using Attributes are:
- In XML 1.0, and in the XML Schema, only attributes may have default values assigned to them by the schema.
- Attributes can have names that indicate the role the value plays in the element. Element contents have content names, but there can be by Attributes only to say what role the content plays in any particular element that contains it.
- Attributes have (minimal) data types.
- Attributes take up less space as there is no need for an end tag. Using attributes for data points results in a drastically smaller document representing the same information.
- Attributes are easier to access in DOM.
- Attributes can be built in are unordered.
- Attributes can be used for data points disambiguates structure and information. Code is much cleaner when using attributes for data points – attributes always contain data points, and elements always contain structure.
- When extracting information from an XML document to store to an RDBMS, or vice-versa, using attributes for data points forms a very clean mapping between the systems - attributes always correspond to columns, while elements always correspond to tables. This makes code to import and export data between RDBMS systems and XML documents easy to write and very flexible.
- Attributes can be constrained against a predefined list of enumerated values.
- Attributes can have default values.
- Attributes are concise and easier to parse than elements.

The disadvantages of using Attributes are:
- Attributes aren't as convenient for long text, large values, or binary entities.
- Attributes can't contain other elements. Therefore, there can't contain nested info.
- Attribute values are harder to search for in search engines
- Attribute values often don't appear on the screen in editing tools (you have to open a special dialog or popup to see them)
- Attribute values can be slightly more awkward to access in processing APIs

- Attributes are ambiguity and not expandable for future changes. Each attribute is either there or not. There is no way to indicate that if you provide this one, you can't provide that one, or if these two are present, then you can't have that one, or if this one is present, then that one has to also be present, and so on
- Whitespace can't be ignored in an attribute value.
- Attributes can only contain multiple values by using tokens.
- Attributes can describe structures in a difficult form by using "xsi:type" only. There is no way to describe a srtucture by using like child elements.
- Attributes are more difficult to manipulate by program code

4   Guidelines

Attributes can actually be used to display of the information what would otherwise be displayed withing the child elements. How can be done a decision when a piece of information is an child-element or an attribute? Tim Bray has written to this proplem: "...when the property has a simple value like a string, we put that in the content of the element; when the property's value is another object, we put a pointer to it in an attribute value and leave the element decribing the property empty."

That solution is one way but a efficient choice for definition depents not on values only. It must be done additionally a consideration of the limitations and special properties of Elements and Attributes which are depending on the disadvantages and advantages of each too.The following considertions may be helpful for using of Elements or Attributes:

- The definition of an Element is advantangeous if the document property relate to the structure of the document.
- An Element should be used to represent a piece of information that can be considered an indpendent object.
- An Element should be used when the information is related via a parent/child relationship to another piece of information. In this case, the element is also a subelement of the element to which it is related.
- An Attribute should be used to represent any information "left over" after defining the objects that have relationships to other objects (and should thus be elements and subelements).
- An Attribute should be used to represent any information that describes other information, such as a status or id.
- An Elemente must be used, if an item needs to occur multiple times, because attributes can have only occur once in an element.
- An Attribut is very useful, if it necessary to limit values to a predefined list, since it is possible to specify a valid list of values for an element.
- Attributes are a better choice, to minimize the file size of target documents.

The following diagram illustrates a way to find out how want to be an Element or an Attributes necessary to be define it. This definition process depends by considering the limitation and special propertiers which are in the following diagram included.

```
                    ┌──────────────────────────────┐
                    ↓                              │
         ╱ Are multiple information ╲              │     ┌───────────────────┐   higher      ⬡ Create an
        ⟨  flat or hierarchical?   ⟩──hierachical─→│ Break information down │──hierachies──→⟨ Groupelement
         ╲                        ╱                │ into flat structure.   │
                    │                              └───────────────────┘
                   flat

         ╱ Are multiple           ╲              ╱ Are the ordered        ╲
        ⟨ information unordered    ⟩──ordered──→⟨ information existing of  ⟩
         ╲ or ordered?            ╱              ╲ two values only?        ╱
                    │                                        │
                unordered                                  yes

                                               ╱ Does a                  ╲
         ╱ Are the unordered      ╲           ⟨  specific value           ⟩
        ⟨ information existing of  ⟩──yes──→  ⟨ represent information about ⟩
         ╲ two values only?       ╱           ⟨ content of same hierachy, ⟩
                    │                          ╲ or is the information    ╱
                   no                           ╲ content itself?        ╱
                                                          │
                                                       meta-data

         ╱ Is each information    ╲           ╱ Does the value have      ╲              ⬡ Create an
        ⟨ to be specified, manipulated,⟩      ⟨ one of an enumeration     ⟩──enumeration─→⟨ Attribute with
        ⟨ organized, consumed by a ⟩──program→⟨ of values or is the value ⟩              ⟨ Enumeration-List
        ⟨ program or by            ⟩           ╲ of free-form?            ╱
         ╲ a human?               ╱
              │        │
            human    free-form

         ╱ Is the content to be   ╲     undesired to be            ⬡ Create an
        ⟨ spell-checked?          ⟩────spell-checked──────────────→⟨ Attribute
         ╲                        ╱
                    │
                    └──────to be spell-checked──────────────────→ ⬡ Create an
                                                                  ⟨ Element
```

5   Recommendation

In the Core Components Technical Specification a Core Component Type will be used for the
creation of Core Components. It consists of one Content Component for the value and one ore
more Suplementary Components for giving an essential extra definition to the Core
Component itself. The Core Component Type will be used for creation of Basic Core
Component (BBC) or Aggregate Core Components (ACC) respectively, which are necessary
for building of Basic Business Information Entities (BBIEs) or Aggregate Business
Information Entities.

Since this BBIEs are a derivation of BCCs and must have a human-readable business semantic definition, the BCCs itself has to be defined as Elements. The content of each Component Content are to be spell-checked in the most of situations. Therefore the Component Content will be represented as an Element-Value.

The Supplementary Components will be represented as Attributes. Since, as the most of the information of each Supplementary Components are restricting attributes, will be used by programs and represented can be represented in a unordered form. Furthermore, the Supplementary Components could be including information as enumerations.

All Aggregate Components (ACCs and ABIEs) are nodes in an hierachical order and nodes inside of hierachies will be defined as Groupelements. The following figure describes the relationship between the Core Components and the Business Information Entities and type of representation in XML-syntax of each component.

Core Component Library
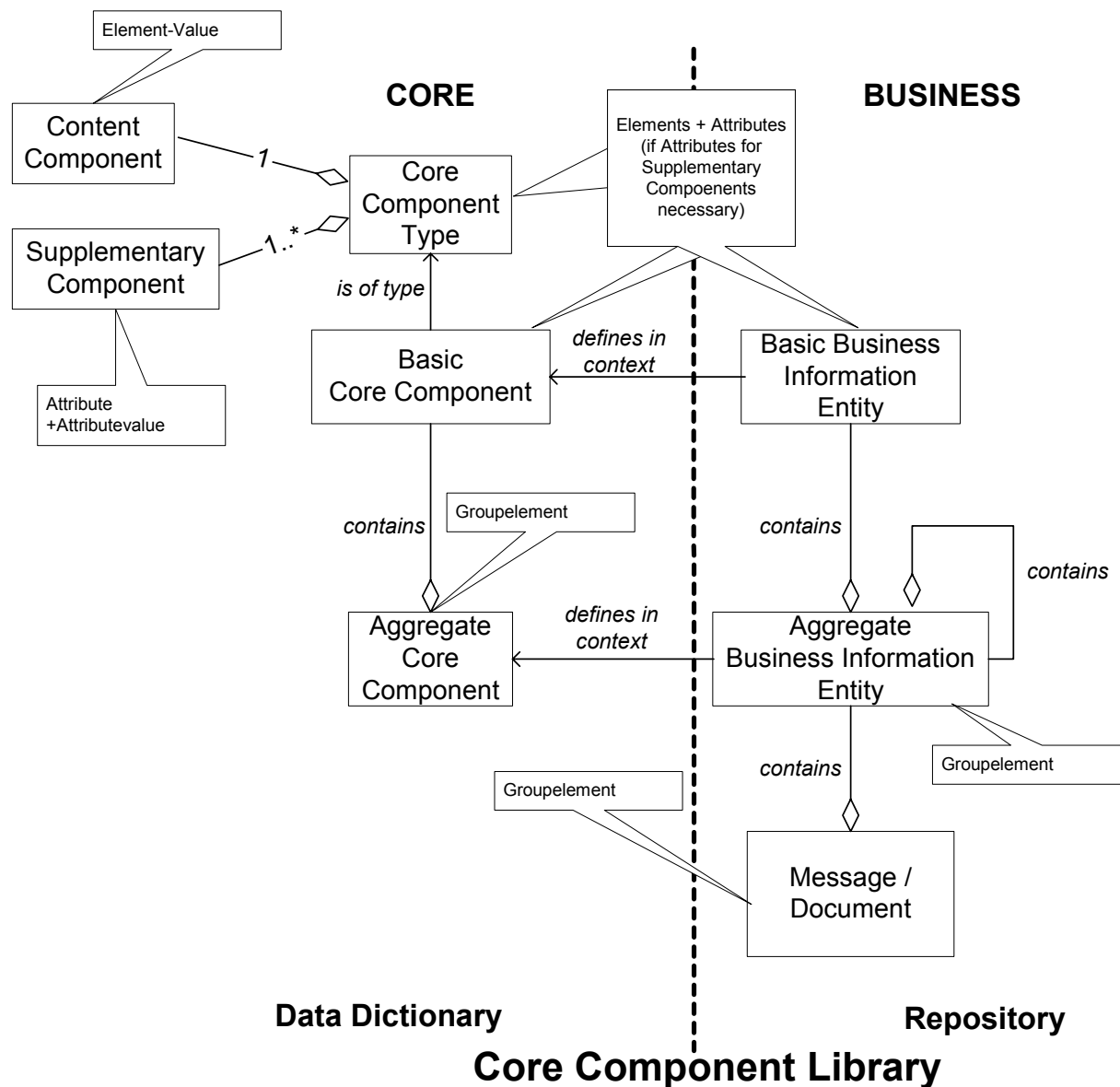
6 Proposal

I would like to do the following proposal for using attributes:

1. Attributes must be used for representation of the Global Unique Identifier by using ID within all components like BCC, ACC, BBIE and ABIE.
2. Attributes should be used for defining the relationship between components. The relationships in XML can also be represented with ID-IDREF(S) attributes. Using these attributes, an element may refer to one or more other elements (by including the value of those elements' ID fields in the pointing element's own IDREF or IDREFS field). While this may seem to be directly analogous to a relational database's key mechanisms, there's one important difference: Most parsers treat these pointers as unidirectional. In other words, given an IDREF or IDREFS field, it's possible to quickly find the element or elements with the associated ID or IDs, but not the other way around. As you'll see when I discuss modeling solutions, this turns out to be a real impediment to design.
3. An attribute should be be used, if the tagname of each CC or BIE will be represented in another language as the Oxford English language. The language Oxford English will be used as default.
4. Attributes should be used only inside of Core Component Types which are used for defining of the Basic Core Components (Leaf Elements).
5. Attributes should be used only for defining of supplementary components only. The supplementary components are fixed defined inside of the ebXML Core Component Specification and must not expanded normally.
6. The content component should be defined only as an element content of each leaf element.
7. An attribute should be not necessary, if it exists a default value for the specific supplementary component.

As the following diagram shows, it will be two types of attributes are necessary:
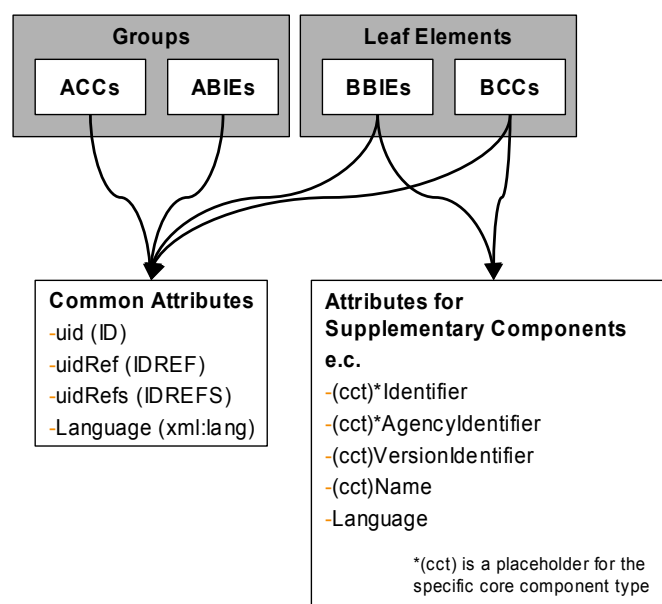
The summary of the properties and the advantages of the proposed way is:

- For each Basic Core Component (BCC) is only one leaf-element necessary. We don't need a element group, which includes a bunch of child elements for the content components as well as for the supplementary components.
- This definition is well structured and easy to read / easy to understand by a user.
- On the other hand the context-dependent BIEs can be easily used in the OO-design and in the implementing coding.
- The information about supplementary components contained in the attribute value only. This attribute values can be omitted in the instances, if the default value is defined.

6.1 Empty Elements

All of the following type of empty elements are not necessary for building Basic Core Components (BCCs) are Basic Business Information Entities (BBIEs) respectively:

```
<ElementName/>

<ElementName></ElementName>

<ElementName attributeName="Value"/>

<ElementName attributeName="Value"></ElementName>
```

Every BBIE derived from a BCC includes a content which is expressed by the element value of a leaf element. Otherwise, it is a content not needed, the specific BBIE must not to be expressed.

6.2 Common Attributes

For the definition of the common attributes (ID, IDREF, IDREFs and language) which will be used within every Core Component and Business Information Entity respectively, there is a attributegroup (the choosen name of that attribute group is "UidAttributeGroup" yet) defined. This attribute group includes the following attributes:

- uid – The attribute "uid" identify each CC or BIE uniquely by expressing the GUID (Global Unique Identifier). The "uid" based on the built-in datatype "ID". The ID must be represented in every CC and BIE.ID represents the ID attribute type from [XML 1.0 (Second Edition)].
- uidRef – The attribute "uidRef" use a single IDREF relationship to point the relating element back to the element it needs to reference. IDREF represents the IDREF attribute type from [XML 1.0 (Second Edition)].
- uidRefs – The attribute "uidRefs" based on the built-in datatype IDREFS. IDREFS can habve serveral targets (IDs). IDREFS represents the IDREFS attribute type from [XML 1.0 (Second Edition)].
- xs:language – The Attribute "language" may be inserted in documents to specify the language used for the tagnames for BIEs and CCs. The attribute represents natural

language identifiers as defined by [RFC 1766]. It will be used, if the tagname of each CC or BIE will be not in the Oxford English language. The language Oxford English will be used as default.

namespace	CoreComponentTypes.xsd					
used by	complexTypes	**AmountType CodeType DateTimeType IdentifierType MeasureType NumericType QuantityType TextType**				
attributes	Name	Type	Use	Default	Fixed	Annotation
	uid	xs:ID	required			
	uidRef	xs:IDREF	optional			
	uidRefs	xs:IDREFS	optional			
	language	language	optional	en		

source	`<attributeGroup name="UidAttributeGroup">`
	 `<attribute name="uid" type="xs:ID" use="required"/>`
	 `<attribute name="uidRef" type="xs:IDREF" use="optional"/>`
	 `<attribute name="uidRefs" type="xs:IDREFS" use="optional"/>`
	 `<attribute name="lang" type="language" use="optional" default="en"/>`
	`</attributeGroup>`					

6.3 Attributes for Supplementary Components

Attributes are useful for supplementary components especially. This will show the first example:

```
<complexType name="AmountType" id="000105">
 <annotation>
  <documentation source="CCTS V1.7" xml:lang="en">
    A number of monetary units specified in a currency where the unit of currency is explicit or implied.
  </documentation>
 </annotation>
 <simpleContent>
  <extension base="cct:AmountContentType">
   <attribute name="amountCurrencyIdentificationCode" type="cct:AmountCurrencyIdentificationCodeType"/>
   <attributeGroup ref="cct:UidAttributeGroup"/>
  </extension>
 </simpleContent>
</complexType>
```

The first example represents the core component type "AmountType". The AmountType is derived by the content component "AmountContentType". Therefore it is possible, that the value of AmountType will represent in the derived Core Component directly and not by an additional child element. And the "AmountType" includes on supplementary component "AmountCurrencyIdentificationCode". This supplementary component is derived by "AmountCurrencyIdentificationCodeType" and is represented as an attribute. The XML instance of this "AmountType" is represented as is follows:

```
<Amount amountCurrencyIdentificationCode="EUR" uid="ID000000" uidRef="ID000000" uidRefs="ID000000 ID000001" language="en">3.14</Amount>
```

The next example shows the "AmountType" without any attribute:

```
<xs:complexType name="AmountType_0p2">
 <xs:sequence>
```

```
  <xs:element name="AmountContent" type="cct:AmountContentType"/>
  <xs:element name="AmountCurrencyIdentificationCode" type="cct:AmountCurrencyIdentificationCodeType"/>
 </xs:sequence>
</xs:complexType>
```

It will be more complicated for the parser as well as the user, if the content will be represented in an additional childelement inside the complexType "AmountType". Therefore it is necessary to create two childelements inside of "AmountType". The XML instance will be then shown as the following example:

```
<Amount_0p2>
 <AmountContent>33.34</AmountContent>
 <AmountCurrencyIdentificationCode>EUR</AmountCurrencyIdentificationCode>
</Amount_0p2>
```

It will be much more data necessary for building an instance of "AmountType". This will be influenced the parsing of big documents especially. As well as that example is not better readable as the first example. The new version of DOM as well as the SAX parser parsing all attributes in a very fast and elegant way, faster as a lot of additional childelements.

The following example shows the creation of date-time elements in two different ways:

The first example shows the definition of the dateTime format with a built-in simpleType:

```
<element name="DateTime_0p3" type="dateTime"/>
```

It is although possible to create the Date Time format in a special format, based on ISO 8601:

```
<complexType name="DateTimeType_0p1">
 <simpleContent>
 <extension base="cct:DateTimeContentType">
  <attribute name="dateTimeFormat" type="cct:DateTimeFormatType"/>
 </extension>
 </simpleContent>
</complexType>
```

The attribute "dateTimeFormat" gives the information about the representation of the special format:

```
<DateTime_0p1 dateTimeFormat="YY-MM-DD">02-02-05</DateTime_0p1>
```

This XML instance represented the content in the same element. Therefore, there is no changing of the representation. That will be not so, if the description of the format will be done by an additional child element:

```
<DateTime_0p2>
 <DateTimeContent>02-02-05</DateTimeContent>
 <DateTimeFormat>YY-MM-DD</DateTimeFormat>
</DateTime_0p2>
```

There are two child elements necessary. One for the content and the other for the format description. That makes much more data and is not so easy understandable as the example before.

There are might be a problem by using more than one supplementary components for one Core Component Type. For example "codeType". Since as the values of each supplementary components do represent some processable data or codes respectively.

There are some examples:

A.)

```
 <Code_0p1 codeListIdentifier="1B" codeListAgencyIdentifier="28" codeListVersionIdentifier="1" codeName="Special Code"
languageCode="en">ABCX</Code_0p1>
```

In the first example (A) are all supplementary components represented as attributes. The problem is that will happen no direct relationship between codeName and languageCode. This must be necessary, because the languageCode is related to the codeName.

B.)

```
<Code_0p2 CodeListAgencyIdentifier="1B" CodeListIdentifier="28" CodeListVersionIdentfier="1">
 <CodeContent>ABCX</CodeContent>
 <CodeName languageCode="en">Special Code</CodeName>
 </Code_0p2>
```

In the second example (B) are the supplementary components shared in attributes and child elements. Supplementary components would like represented as attributes, if the data could be processably or coded information respectively. Supplementary components which represents user readable information represented as child elements. The content component is represented as child element, too. One expection have the attribute "languageCode" due to related to the readable name of the code it will be placed inside of the child element "CodeName".

C.)

```
<Code_0p3>
 <CodeContent>ABCX</CodeContent>
 <CodeListAgencyIdentifier>1B</CodeListAgencyIdentifier>
 <CodeListIdentifier>28</CodeListIdentifier>
 <CodeListVersionIdentifier>1</CodeListVersionIdentifier>
 <CodeName>Special Code</CodeName>
 <LanguageCode>us</LanguageCode>
 </Code_0p3>
```

The last example the CodeType without any attributes. There are much more data and there is no relationship between CodeName and LanguageCode, too.
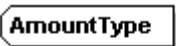
The attributes doesn't make the readability much more complicated. It help us, to build relationship in a very short and easy matter. The XML instances are much shorter and there

are not so much hierachies for representing that data. That helps that the parsing of that structure is much more faster. And is helpful to map elements in an internal workflow or database respectively.
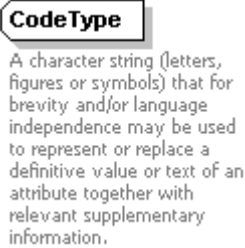
6.4 Attributes within the Core Component Types

The following subchapters shows the different core component types with the use of attributes as examples.

6.4.1   complexType AmountType

diagram	**AmountType** A number of monetary units specified in a currency where the unit of currency is explicit or implied.					
namespace	CoreComponentTypes.xsd					
type	extension of **cct:AmountContentType**					
facets	totalDigits 10 fractionDigits 2					
attributes	Name	Type	Use	Default	Fixed	Annotation
	amountCurrency IdentificationCod e	cct:AmountCurr encyIdentificatio nCodeType				
	uid	xs:ID	required			
	uidRef	xs:IDREF	optional			
	uidRefs	xs:IDREFS	optional			
	language	language	optional	en		
annotation	documentation	A number of monetary units specified in a currency where the unit of currency is explicit or implied.				

source	`<complexType name="AmountType" id="000105">`
	 `<annotation>`
	  `<documentation source="CCTS V1.7" xml:lang="en">`A number of monetary units specified in a currency where the unit of currency is explicit or implied.`</documentation>`
	 `</annotation>`
	 `<simpleContent>`
	  `<extension base="cct:AmountContentType">`
	   `<attribute name="amountCurrencyIdentificationCode" type="cct:AmountCurrencyIdentificationCodeType"/>`
	   `<attributeGroup ref="cct:UidAttributeGroup"/>`
	  `</extension>`
	 `</simpleContent>`
	`</complexType>`					

example	`<Amount amountCurrencyIdentificationCode="EUR" uid="ID000000" uidRef="ID000000" uidRefs="ID000000 ID000001" language="en">3.14</Amount>`					

6.4.2 complexType CodeType

diagram	**CodeType** A character string (letters, figures or symbols) that for brevity and/or language independence may be used to represent or replace a definitive value or text of an attribute together with relevant supplementary information.
namespace	CoreComponentTypes.xsd
type	extension of **cct:CodeContentType**

attributes	Name	Type	Use	Default	Fixed	Annotation
	codeListIdentifier	cct:CodeListIdentifierType				
	codeListAgencyIdentifier	cct:CodeListAgencyIdentifierType				
	codeListVersionIdentifier	cct:CodeListVersionIdentifierType				
	codeName	cct:CodeNameType				
	languageCode	cct:LanguageCodeType				
	uid	xs:ID	required			
	uidRef	xs:IDREF	optional			
	uidRefs	xs:IDREFS	optional			
	language	language	optional	en		

annotation	documentation	A character string (letters, figures or symbols) that for brevity and/or language independence may be used to represent or replace a definitive value or text of an attribute together with relevant supplementary information.

source	`<complexType name="CodeType" id="000089">`
	 `<annotation>`
	  `<documentation source="CCTS V1.7" xml:lang="en">A character string (letters, figures or symbols) that for brevity and/or language independence may be used to represent or replace a definitive value or text of an attribute together with relevant supplementary information.</documentation>`
	 `</annotation>`
	 `<simpleContent>`
	  `<extension base="cct:CodeContentType">`
	   `<attribute name="codeListIdentifier" type="cct:CodeListIdentifierType"/>`
	   `<attribute name="codeListAgencyIdentifier" type="cct:CodeListAgencyIdentifierType"/>`
	   `<attribute name="codeListVersionIdentifier" type="cct:CodeListVersionIdentifierType"/>`
	   `<attribute name="codeName" type="cct:CodeNameType"/>`
	   `<attribute name="languageCode" type="cct:LanguageCodeType"/>`
	   `<attributeGroup ref="cct:UidAttributeGroup"/>`
	  `</extension>`
	 `</simpleContent>`
	`</complexType>`

example	`<Code codeListIdentifier="CODEID01" codeListAgencyIdentifier="CodeAgency" codeListVersionIdentifier="V01" codeName="CodeName" languageCode="en-us" uid="ID000001" uidRef="ID000001" uidRefs="ID000000 ID000001" language="en">COD</Code>`

6.4.3 complexType DateTimeType

diagram	DateTimeType A particular point in the progression of time together with relevant supplementary information. Can be used for a date and/or time.					
namespace	CoreComponentTypes.xsd					
type	extension of **cct:DateTimeContentType**					
attributes	Name	Type	Use	Default	Fixed	Annotation
	dateTimeFormat Text	cct:DateTimeFor matTextType				
	uid	xs:ID	required			
	uidRef	xs:IDREF	optional			
	uidRefs	xs:IDREFS	optional			
	language	language	optional	en		
annotation	documentation	A particular point in the progression of time together with relevant supplementary information. Can be used for a date and/or time.				

source	`<complexType name="DateTimeType" id="000066">`
	`<annotation>`
	`<documentation source="CCTS V1.7" xml:lang="en">`A particular point in the progression of time together with relevant supplementary information.
	Can be used for a date and/or time.
	`</documentation>`
	`</annotation>`
	`<simpleContent>`
	`<extension base="cct:DateTimeContentType">`
	`<attribute name="dateTimeFormatText" type="cct:DateTimeFormatTextType"/>`
	`<attributeGroup ref="cct:UidAttributeGroup"/>`
	`</extension>`
	`</simpleContent>`
	`</complexType>`					

example	`<DateTime dateTimeFormatText="YYYY-MM-DD" uid="ID000002" uidRef="ID000002" uidRefs="ID000001 ID000002" language="en">`2002-03-05`</DateTime>`					

6.4.4 complexType IdentifierType
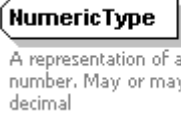
diagram	IdentifierType A character string to identify and distinguish uniquely, one instance of an object in an identification scheme from all other objects within the same scheme together with relevant supplementary information.					
namespace	CoreComponentTypes.xsd					
type	extension of **cct:IdentifierContentType**					
attributes	Name	Type	Use	Default	Fixed	Annotation
	identificationSch emeName	cct:Identification SchemeNameTy pe				
	iIdentificationSc hemeAgencyNa me	cct:Identification SchemeAgency NameType				
	languageCode	cct:LanguageCo				

		deType			
	uid	xs:ID	required		
	uidRef	xs:IDREF	optional		
	uidRefs	xs:IDREFS	optional		
	language	language	optional	en	
annotation	documentation	A character string to identify and distinguish uniquely, one instance of an object in an identification scheme from all other objects within the same scheme together with relevant supplementary information.			

source	`<complexType name="IdentifierType" id="000101">`
	`<annotation>`
	`<documentation source="CCTS V1.7" xml:lang="en">A character string to identify and distinguish uniquely, one instance of an object in an identification scheme from all other objects within the same scheme together with relevant supplementary information. </documentation>`
	`</annotation>`
	`<simpleContent>`
	`<extension base="cct:IdentifierContentType">`
	`<attribute name="identificationSchemeName" type="cct:IdentificationSchemeNameType"/>`
	`<attribute name="iIdentificationSchemeAgencyName" type="cct:IdentificationSchemeAgencyNameType"/>`
	`<attribute name="languageCode" type="cct:LanguageCodeType"/>`
	`<attributeGroup ref="cct:UidAttributeGroup"/>`
	`</extension>`
	`</simpleContent>`
	`</complexType>`				

example	`<Identifier identificationSchemeName="IDNAME01" iIdentificationSchemeAgencyName="IdAgency" languageCode="en-us" uid="ID000003" uidRef="ID000003" uidRefs="ID000002 ID000003" language="en">ID01022-XX</Identifier>`				

6.4.5 complexType MeasureType

diagram	 **MeasureType** The size, volume, mass, amount or scope derived by performing a physical measure together with relevant supplementary information.
namespace	CoreComponentTypes.xsd
type	extension of **cct:MeasureContentType**

attributes	Name	Type	Use	Default	Fixed	Annotation
	measureUnitCode	cct:MeasureUnitCodeType				
	uid	xs:ID	required			
	uidRef	xs:IDREF	optional			
	uidRefs	xs:IDREFS	optional			
	language	language	optional	en		

annotation	documentation	The size, volume, mass, amount or scope derived by performing a physical measure together with relevant supplementary information.

source	`<complexType name="MeasureType" id="000152">`
	`<annotation>`
	`<documentation source="CCTS V1.7" xml:lang="en">The size, volume, mass, amount or scope derived by performing a physical measure together with relevant supplementary information.</documentation>`
	`</annotation>`
	`<simpleContent>`
	`<extension base="cct:MeasureContentType">`
	`<attribute name="measureUnitCode" type="cct:MeasureUnitCodeType"/>`
	`<attributeGroup ref="cct:UidAttributeGroup"/>`
	`</extension>`
	`</simpleContent>`
	`</complexType>`	

example	`<Measure measureUnitCode="KGM" uid="ID000004" uidRef="ID000004" uidRefs="ID000003 ID000004" language="en">3.14</Measure>`	

6.4.6 complexType NumericType

diagram	NumericType A representation of a number. May or may not be decimal
namespace	CoreComponentTypes.xsd
type	extension of **cct:NumericContentType**

attributes	Name	Type	Use	Default	Fixed	Annotation
	numericFormatTextType	cct:NumericFormatTextType				
	uid	xs:ID	required			
	uidRef	xs:IDREF	optional			
	uidRefs	xs:IDREFS	optional			
	language	language	optional	en		

annotation	documentation A representation of a number. May or may not be decimal

| source | ```xml
<complexType name="NumericType" id="000182">
  <annotation>
    <documentation source="CCTS V1.7" xml:lang="en">A representation of a number. May or may not be decimal</documentation>
  </annotation>
  <simpleContent>
    <extension base="cct:NumericContentType">
      <attribute name="numericFormatTextType" type="cct:NumericFormatTextType"/>
      <attributeGroup ref="cct:UidAttributeGroup"/>
    </extension>
  </simpleContent>
</complexType>
``` |
|---|---|

example	`<Numeric numericFormatTextType="nnnnnn" uid="ID000005" uidRef="ID000005" uidRefs="ID000004 ID000005" language="en">123324</Numeric>`

6.4.7 complexType QuantityType

diagram	QuantityType A number of non-monetary units together with relevant supplementary information.
namespace	CoreComponentTypes.xsd
type	extension of **cct:QuantityContentType**

attributes	Name	Type	Use	Default	Fixed	Annotation
	quantityUnitCode	cct:QuantityUnitCodeListAgencyIdentifierType				
	quantityUnitCodeListIdentifier	cct:QuantityUnitCodeListIdentifierType				
	quantityUnitCodeListAgencyIdentifer	cct:QuantityUnitCodeListAgencyIdentifierType				
	uid	xs:ID	required			
	uidRef	xs:IDREF	optional			
	uidRefs	xs:IDREFS	optional			
	language	language	optional	en		

annotation	documentation A number of non-monetary units together with relevant supplementary information.

source	`<complexType name="QuantityType" id="000108">`
	 `<annotation>`
	  `<documentation source="CCTS V1.7" xml:lang="en">A number of non-monetary units together with relevant supplementary information.</documentation>`
	 `</annotation>`
	 `<simpleContent>`
	  `<extension base="cct:QuantityContentType">`
	   `<attribute name="quantityUnitCode" type="cct:QuantityUnitCodeListAgencyIdentifierType"/>`
	   `<attribute name="quantityUnitCodeListIdentifier" type="cct:QuantityUnitCodeListIdentifierType"/>`
	   `<attribute name="quantityUnitCodeListAgencyIdentifer" type="cct:QuantityUnitCodeListAgencyIdentifierType"/>`
	   `<attributeGroup ref="cct:UidAttributeGroup"/>`
	  `</extension>`
	 `</simpleContent>`
	`</complexType>`

example	`<Quantity quantityUnitCode="token" quantityUnitCodeListIdentifier="token" quantityUnitCodeListAgencyIdentifer="token" uid="ID000006" uidRef="ID000006" uidRefs="ID000005 ID000006" language="en">10</Quantity>`

6.4.8 complexType TextType

diagram	**TextType** A character string with or without a specified language.
namespace	CoreComponentTypes.xsd
type	extension of **cct:TextContentType**
attributes	

Name	Type	Use	Default	Fixed	Annotation
languageCode	cct:LanguageCodeType				
uid	xs:ID	required			
uidRef	xs:IDREF	optional			
uidRefs	xs:IDREFS	optional			
language	language	optional	en		
language	xs:language	optional			

annotation	documentation A character string with or without a specified language.

source	`<complexType name="TextType" id="000090">`
	 `<annotation>`
	  `<documentation source="CCTS V1.7" xml:lang="en">A character string with or without a specified language.</documentation>`
	 `</annotation>`
	 `<simpleContent>`
	  `<extension base="cct:TextContentType">`
	   `<attribute name="languageCode" type="cct:LanguageCodeType"/>`
	   `<attributeGroup ref="cct:UidAttributeGroup"/>`
	   `<attribute name="language" type="xs:language" use="optional"/>`
	  `</extension>`
	 `</simpleContent>`
	`</complexType>`

example	`<Text languageCode="en-us" uid="ID000007" uidRef="ID000007" uidRefs="ID000006 ID000007" language="en" lang="en-us">Hello World</Text>`