Project Number	IST-2006-033789
Project Title	Planets
Title of Deliverable	Final XCDL Specification
Deliverable Number	D7
Contributing Sub-project and Work-package	PC/2
Deliverable Dissemination Level	External Public
Deliverable Nature	Report
Contractual Delivery Date	31st May 2008
Actual Delivery Date	31st May 2008
Author(s)	HKI – University at Cologne

Abstract

This document describes the status of the XCDL development in May 2008. It contains a reference to the language, but also an introduction into the creation (and interpretation) of XCDL-documents. Both parts should also enable the reader to develop XCDL-aware software.

Keyword list

XCDL, XCL, XCEL, PC, Extensible Characterisation Description Language

Contributors

Person	Role	Partner	Contribution
Sebastian Beyl	Author	UzK	
Volker Heydegger	Contributor	UzK	Common coactions
Jan Schnasse	Contributor	UzK	Common coactions
Manfred Thaller	Contributor	UzK	Section 5, modelling information

Document Approval

Person	Role	Partner
Hannes Kulovits	Reviewer	TUWIEN

Distribution

Person	Role	Partner

Revision History

Issue	Author	Date	Description
1.0	Sebastian Beyl	31th May 2008	Original reference documentation
1.1	Manfred Thaller	2nd July 2008	Minor modifications after internal review

References

Ref.	Document	Date	Details and Version

EXECUTIVE SUMMARY

The Extensible Characterisation Description language (XCDL) allows the representation of characteristics extracted from files. The definition of characteristics, however, is taken in the broadest possible way: So, conceptually, an XCDL representation of the information contained within a file can be a complete interpretation of all the information contained in that file. While some of the conceptual implications of this approach are described in this document, presenting the XCDL as a language for multi-purpose representation of information is not its primary purpose. That is a description of the XCDL for the purpose it serves within the Planets project: The representation of the way in which information from one digital object is represented within different file formats, to allow their comparison during the evaluation of migrations and similar preservation actions.

TABLE OF CONTENTS

1. Introduction

Relationship of this version of the XCDL to the original specification

The XCDL has been developed to describe a large variety of types of file content. XML was used as the background-technology. Digital objects can split up into their components and their content can be saved, represented in the XCD language. To integrate the language smoothly into the developing world of standards, it is based upon an xml-schema. This allows standard XCDL documents to be validated without creating new tools.

While splitting the information contained in a file into the categories of its XCDL representation, we differentiate between “raw” information, so called “normalised data” or “normdata” and the description of that information, so called ”properties”. The XCDL representation of a file consists of one or more normdata and properties, which relate to these normdata and define their meaning, together describing the complete content of the file. XCDL descriptions are always directly related to files and when we discuss the details of the language, they all pertain to byte and bit sequences. We will see, however, that any one XCDL description may be related to digital content which is distributed over more than one file; and there are cases, where the content of a physical file is structured in such a way, that the XCDL describing it is actually subdivided into a number of XCDL descriptions, which are almost completely independent of each other. To mark this difference, we will say in this document that the XCDL may describe digital *objects*.

This is the reason for the main differences between the structure of the XCDL as described in the first version of the specification and the specification we present here. In the original version, the design of the XCDL was very much influenced by the assumption that for *one* physical file *one* distinct XCDL description would be produced.

The current, final specification of the XCDL has been built around an extended understanding of this relationship: On the one hand, more recent file formats frequently describe a structure, where a number of physically distinct files are grouped together by all software handling it into one container, which exists for purposes of data storage and transfer, is unpacked into individual files for processing however. The meaningful unit of information in such a case is the container, where the information is stored – and can therefore be preserved; not the individual physical files into which that information is distributed transiently for ease of processing. In that sense, therefore, the XCDL description of a permanent file may describe, what transiently is a group of files.

On the other hand, for reasons which are described in more detail in chapter 5 of this document, the XCDL assumes, that an XCDL document typically describes some data, which belong to a specific type of data, like an image, a text, a streaming medium etc. This principle has never hindered to understand, that one such document may contain data of another type, as, e.g., the textual string containing the copyright information within an image. It has turned out to be convenient, however, to provide for a more general purpose, by which there is a clear interface between data objects of different types for cases where, e.g., a XCDL description of a text contains an XCDL description of an image.

The changes between the first and the current, final, version of the XCDL reflect the consequences of these two problems. The “composition element” of the original XCDL specification has been dropped for a more general mechanism to describe the inclusion of XCDLs of one type into XCDLs of another; to ease the description of embedded objects the original specification of “objRel” nodes has been replaced by an “objectRef” node, which also shows the direction of the relationship.

The exact specification of the XCDL - and examples for the use of the features thus specified - are presented in the following sections. The specification presented here is “final” in the sense that it has the capability to map the content of files in rather complex formats – PDF and OOXML. These have been sufficiently different from the image formats, which have been the main domain upon which the first version of the language has been built, to lead to the significant changes mentioned above. While we assume that the intensified use of the XCDL to describe the content of additional types of files in the future will lead to fewer and less significant changes, even a “final” language will have to develop further.

Main Purpose of the XCDL

The XCDL has been formulated as a means to give an abstract definition of the content of a file or a set of files. This is a broad and general goal. It has been pursued, however, not as an abstract exercise but with an immediate practical purpose in mind, which unavoidably has influenced many of the individual decisions. XCDL describes the content of digital files. These descriptions are – within Planets - used to compare two digital representations of one object in two different file formats in a standardized way. The XCDL is supposed to support and optimize these automatic comparisons.

These comparisons are performed by a software tool called “comparator”. It should not only do a comparison as a binary “diff” tool, which answers the question whether some digital objects are identical with “true or false, ” but it should give a degree of similarity. This makes it possible to migrate large archives from an old to a new format, control the overall success of that migration and also calculate the loss of information during migration. The “comparator” calculates this information loss by matching the properties and normData from a source file to the corresponding properties and normData of a target file. The summarized degrees of equality between of the paired normData and properties within the two files define the equality of source and target file.

The XCDL has not been designed as a data format to archive data. We acknowledge, however, that the idea to convert the content of a file encoded in a contemporary format into a XCDL document, preserve that document, and reconvert it later into a contemporary format of that later day, would be a logical extension of the XCDL approach. It will not be pursued within Planets; no precautions have been considered to make the language more useful for such an approach. On the other hand, all structures have been designed to make the comparisons of XCDL documents as easy as possible, even if that may have made some structures more complex than they would have been, if only the representation of the data on an abstract level would have been attempted.

One, though not the only one, reason for that is, that within Planets the XCDL is supposed to be *able* to represent 100 % of the information contained within a file which it represents, it is not *required*, however, that it is used in that way. Indeed, the extractor based upon the current definition of XCDL’s sister language used to describe file formats, implicitly encourages to extract only parts of the information contained within a file. This is done on the one hand, to be able to process highly complex file formats, where a complete translation of the file format into XCEL will take time (PDF, e.g.); on the other hand this shall encourage using the XCDL / XCEL approach also to handle file formats which are only partially understood.

2. The XCDL-Specification and Examples

As already mentioned, the XCDL has been created to allow the representation of as many digital objects as possible in a standardized way, based upon XML. The formal specification will be illustrated by small examples showing the usage of the features specified, in the spirit of the introduction of a language by its handling of “hello, world!”.

A First XCDL Document

As first simple introduction to the XCDL, we like to show how to represent a very simple digital object in the language. Knowing the basis of the technique will help to understand more complex usages of the XCDL.

The following text is going to be represented by a XCDL document:

```
This is only a short text.
```

There is no further description of the text, no fonts or other format information is indicated or specified. Therefore a short XCDL document will look the following way:

```
<?xml version='1.0' encoding='utf-8'?>
<xcdl xmlns:xsi="http://www.planets-project.eu/xcl/schemas/xcl"
xsi:schemaLocation="http://www.planets-
project.eu/xcl/schemas/xcl/XCDLCore.xsd" id="0">
  <object id="o1">
    <normData id="nd1" type="text">This is only a short text.</normData>
  </object>
</xcdl>
```

As you see, the text is included as a normData node. In turn the normData are surrounded by an object-node. This indicates that this is a self-contained digital unit. Within this unit no more information is specified, as the text to be handled did, as mentioned, not contain any formatting specification, although this is quite unusual for a text format.

Let’s look at the individual XML tags, or nodes, in the XCDL representation:

xcdl (XML-Node)

The xcdl-node is always the root-element of an XCDL-document, but not all digital objects must necessarily be described by a single xcdl node: If more than one XCDL-document describes a digital object, each of them have an xcdl-node as the root-node.

This is a child of node:

`none` (root-element)

Attributes:

`id` - The attribute `id` is an unique identifier for the xcdl-node and required. Its value is string or an integer, like all the identifiers of xml-elements of the XCDL we encounter in this document.

Objects

As already shown in the first example, all XCDL represented data are structured as objects. Those objects represent logical entities. There can be more than one of these logical entities within a digital document.

It is not always clear, how to separate two digital objects. Some structures can only be described with the help of a few objects and references between them. XCDL only knows normData and properties. Properties can be related to normData. But XCDL has no concept of properties directly related to other properties. For that reason it is necessary to describe, e.g., footnotes as an independent object. Footnotes can be related to normData, will themselves consist of their own normData, however, and the properties, describing them.

As it is necessary to establish a relationship between these different objects, the XML-node `object` possesses the attribute `id`. The value of this attribute has to be unique within an XCDL document.

If you establish a relationship between two objects in this way, one object becomes logically a property of another.

Later we will give a more detailed explanation of property structures. At the moment lets have a closer look at the XML-node `objectRef` . A text with a footnote is described in an XCDL document as follows:

```
This is only a short1 text.


----------------

1: short is another term for "not long".
```

```
<?xml version='1.0' encoding='utf-8'?>
<xcdl xmlns:xsi="http://www.planets-project.eu/xcl/schemas/xcl"
xsi:schemaLocation="http://www.planets-
project.eu/xcl/schemas/xcl/XCDLCore.xsd" id="0">
  <object id="o1">
    <normData id="nd1" type="text">This is only a short1 text.</normData>
    <property id="p1" source="raw" cat="descr">
      <name id="id170">footnote</name>
      <valueSet id="vs1">
        <objectRef>.:02</objectRef>
      </valueSet>
  </object>
  <object id="o2">
    <normData id="nd2" type="text">1: short is another term for "not
long".</normData>
  </object>
</xcdl>
```

This document can not be validate against the XML-Schema of the XCDL, as, for clarity's sake some required elements are absent in this example; these will be explained later. But even in the simplified version you can see that there is a property within the first object with the identifier `o1`, which is called `footnote`. Furthermore there exists a node `objectRef` within this property. This refers to the object `o2`. This object is shown later in the file.

The notation `.:o2` describes that there is an object with the identifier o2 to be found in this file. Another file can be specified like this: `file://otherFile.xcdl:o2`. In this case the object with identifier o2 would be found in the file `otherFile.xcdl`.

Such assemblies of properties will be explained later in detail, as will be the usage of the `valueSet`. There is an important point to note about this kind of connection of objects. The function of the linked object is not defined by this object itself but by the property to which it is assigned, in this case called 'footnote'. This separation between the definition of an object and the interpretation of its usage is also the reason why an object can be used several times within an XCDL document, each time in a different function.

Object (XML-Node)

The XML-node 'object' represents a logical information unit. These units are built by reading and interpreting the original files. In any one XCDL document there can be may objects.

This is a child of the node:

`xcdl`

Attributes:

`id` - The attribute `id` is an unique identifier for the `xcdl`-node and required. Its value is string or an integer, like all the identifiers of xml-elements of the XCDL we encounter in this document.

objectRef (XML-Node)

The xml-node objectRef references another object, contained within the same or another XCDL document. The string contained within the <objectRef> Tag specifies the location of the related object.

This is a child of the node:

`valueSet`

Attributes:

none

Syntax of the reference within the tag:

A path to the related object, which follows the syntax: `(file://.*|\.):.*` The first describes a protocol, in our case always file://, the second fragment is a file or a period point for the file currently being processed, the third fragment is the `id` for the related object. So "`file://otherFile.xcdl:o2`" stands for the object "o2" within the xcdl document "otherFile.xcdl". "`.:o2`" would refer to object "o2" within the current file.

normData

As already shown in the first example normData contain basic information derived form a file, which is specified further by properties. They do not change, when only the properties describing them change. The following text is another short example:

```
This is only a short text.
```

The text occurs in normData like this:

```
<?xml version='1.0' encoding='utf-8'?>
<xcdl xmlns:xsi="http://www.planets-project.eu/xcl/schemas/xcl"
xsi:schemaLocation="http://www.planets-
project.eu/xcl/schemas/xcl/XCDLCore.xsd" id="0">
  <object id="o1">
    <normData id="nd1" type="text">This is only a short text.</normData>
  </object>
</xcdl>
```

If you change properties of the text, it might look like this:

This is *only* a **short** text.

The normData are not affected by this change of properties. The XCDL representation if the normData remains unchanged, even though some aspects of the text have obviously changed. The same principle of separation between a basic stream of data, contained within the normData and their properties applies for pictures. The normData in this case contain the uninterpreted values of each individual pixel.

It is not always easy to differentiate between normData and their properties. If a word within a text has been crossed out, the meaning, to be communicated by the text, has changed drastically. That part of the text has been crossed out, will nevertheless not be reflected by the normData. Even crossed out text is considered as and will be treated as text. Its representation, described by an appropriate property, will not influence, whether it is part of the normData or not.

Similarly footnotes - from the previous example - create an interpretative problem. A footnote could appear like this in a source format, that shall be transformed into XCDL:

```
<text>This is a new<footnote>new is related to an earlier
document</footnote> text.</text>
```

This rendered text in an editor will look like this:

```
This is a new1 text.

-----------------------

1: new is related to an earlier document
```

Indeed the footnote can be shown like this:

```
This is a new1 text.

-----------------------

1- new is related to an earlier document
```

There was only a change of the separator between the character 1 and the text. But is this text still a part of the normData? Not necessarily: If the character separating footnote counter and footnote text is clearly defined as an abstract property, of the text as a whole, by the format specification, it will not be considered part of the normData.

In the same way the source file could contain:

```
This is a new2 text.

-----------------------

2- new is related to an earlier document
```

The counter of the footnote changed to 2, presumably because another footnote was added in an earlier part of the text. The same could, however, also have happened, because by the selection of another text-style or another paper size, a footnote happened to be transferred from one page to another during layout. In cases, where the layout is determined by the software formatting the text, and resulting details like the footnote counter used are not contained within the permanent part of a textual file, these details will not become part of the normData.

normData (XML-Node)

The <normData> element inside the XML-Node contains the basic, uninterpreted data of a file. For text files, that is a sequence of UTF-8 characters; for images it is a sequence of byte values. They reflect no interpretation of such uninterpreted data by the software which is interpreting the file format from which a XCDL document containing these normData has been derived.

This is a child of the node:

```
object
```

Attributes:

`id` - This is the mandatory identifier of the normData node which *must* be unique within the object containing the normData node, and *should* be unique within the XCDL-document, containing the document.

`type` - This attribute is required and describes the basic type of information within the text-node of the normData-Node. Only the following types are allowed: `text`, `image`, `audio`, `object` and `other`. For current purposes the first four of these types are sufficient; if support of other basic types will be added, we propose rather more to add additional appropriate types than using “other” indiscriminately.

Properties

The plain text of a text document or the pixel values of an image are not sufficient to describe the contents of a file containing such information. The information about how these basic data are to be interpreted has also to be extracted. XCDL solves this problem with the aid of *properties*.

A XCDL *property* is the description of the value an abstract characteristic of some data takes within a specific file. We could also say, that properties describe which values of a potential characteristic shall be applied to which part of the uninterpreted data. We can demonstrate this by looking at the representation of the fact that part of a text is to be displayed in italics:

`This is a sentence with` *`a few italic`* `words.`

Leaving aside the normData for the moment, the property '`italic`' appears within the XCDL like this:

```
<?xml version='1.0' encoding='utf-8'?>
<xcdl xmlns:xsi="http://www.planets-project.eu/xcl/schemas/xcl"
xsi:schemaLocation="http://www.planets-
project.eu/xcl/schemas/xcl/XCDLCore.xsd" id="0">
  <object id="o1">
    <property id="p1">
      <name id="id162">italic</name>
    </property>
  </object>
</xcdl>
```

Inside of the object `o1` we have now property with its identifier exists, in this case `p1`. This representation is not very useful so far, as it just says that something in the file described by this XCDL fragment is in italics, without specifying what that something is. To solve this and connect the property 'italic' to a specific section of the normData, we use:

```
<?xml version='1.0' encoding='utf-8'?>
<xcdl xmlns:xsi="http://www.planets-project.eu/xcl/schemas/xcl"
xsi:schemaLocation="http://www.planets-
project.eu/xcl/schemas/xcl/XCDLCore.xsd" id="0">
  <object id="o1">
```

```
    <normData id="nd1" type="text">This is a sentence with a few italic
words.</normData>
    <property id="p1">
      <name id="id162">italic</name>
      <valueSet id="vs1">
        <labValue>
          <val>default</val>
          <type>XCLLabel</type>
        </labValue>
        <dataRef ind=”normSpecific” propertySetId=”ps1” />
      </valueSet>
    </property>
    <propertySet id="ps1">
      <valueSetRelations>
        <ref valueSetId=”vs1” />
      </valueSetRelations>
      <dataRef>
        <ref begin=”24” end=”35” id=”nd1” />
      </dataRef>
    </propertySet>
  </object>
</xcdl>
```

In this complete example, the normData are back, our sentence is now included a normData- node node of type 'text' identified as `nd1`. Second, the property has been augmented by an XML-node called `valueSet`. At this point, it is not important why we do this, we will explain it later in this chapter. More important is the `dataRef`-Node, which contains one or two attributes. If the attribute `ind` is set to the value '`global'`, the valueSet and with it the parent-node property will be linked to the whole normData of this object. If, for example, the whole document is written in italic, you will have made this clear by using the attribute `ind` with the value `global`.

If you have to apply the property to a short sequence of the normData only, it is a little bit more complicated. In an earlier version of the XCDL you were allowed to use offsets into the content of the normData indicate the points where the application of a property should start and stop. And the conceptual notion, that properties relate *from* a given offset within the normData *to* another one, is still useful. To understand what is happening, you can simply glimpes at the <dataRef> element and assume, that its 'begin' and 'end' attributes define to which part of the normData string the property in question has to be applied. While this solution would be intuitive, it leads, unfortunately to problems with more complex formats.

Therefore you are required now to bind a property first into a propertySet and connect only this propertySet to a part of the normData. The propertySet is simply created by opening a XML-node called propertySet, identified as ps1 in this example. As you can see in the dataRef-structure of the valueSet, which we mentioned already, this identification is used to link from a valueSet to a propertySet. Reversely, the propertySet also links to the valueSet itself, opening a valueSet-Node and collecting all used valueSets by storing their identifiers in an attribute valueSetId inside a XML-Node ref. With this technique, you are able to define a set of properties within a logical unit.

Finally, you have to connect this collection to one or more normData-sections or parts of such. For this, you have to create the XML-node `dataRef`, in which you can now collect `ref`-nodes. These ref-nodes, in our example only one, have to link to a normData-identifier and a beginning and an ending offset. The offset is a simple count of characters within the normData starting with `0` for the first sign.

Having put all together, we now have a giant construct for just one single property 'italic', applied to just one short sequence of characters within the normData-string. Please remember that the XCDL is not designed for being primarily humanly readable, but is assumed to be processed by software. And, in the next sections we will show, that this way of handling properties can easily be generalised for considerably more complex constructs.

As indicated, the next step will show you, why we are using the valueSet-nodes and not binding the property to a propertySet directly. The next example will make this clearer. In it, there is only one word in our normData, but as you can see, many font sizes are used.

`oneWordButManyFontSizes`

The font size grows up from 12 points to 22 points. But though there are six different font sizes, these are only six occurrences of one and the same property. So every valueSet in the next example stands for one occurrence of the property `fontSize`.

```
<?xml version='1.0' encoding='UTF-8'?>
<xcdl xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://www.planets-project.eu/xcl/schemas/xcl"
xsi:schemaLocation="http://www.planets-project.eu/xcl/schemas/xcl
XCDLCore.xsd" id="0" >
  <object id=”o1”>
    <normData id="nd1" type="text">OneWordButManyFontSizes</normData>
    <property id=”p1” source=”raw” cat=”descr”>
name id="id158">fontSize</name>
      <valueSet id=”vs1”>
        <labValue>
          <val unit=”point”>12</val>
          <type>rational</type>
        </labValue>
        <dataRef ind=”normSpecific” propertySetId=”ps1” />
      </valueSet>
      <valueSet id=”vs2”>
        <labValue>
          <val unit=”point”>14</val>
          <type>rational</type>
        </labValue>
        <dataRef ind=”normSpecific” propertySetId=”ps2” />
      </valueSet>
      <valueSet id=”vs3”>
        <labValue>
          <val unit=”point”>16</val>
          <type>rational</type>
        </labValue>
        <dataRef ind=”normSpecific” propertySetId=”ps3” />
      </valueSet>
      <valueSet id=”vs4”>
        <labValue>
          <val unit=”point”>18</val>
          <type>rational</type>
        </labValue>
        <dataRef ind=”normSpecific” propertySetId=”ps4” />
      </valueSet>
      <valueSet id=”vs5”>
        <labValue>
          <val unit=”point”>20</val>
          <type>rational</type>
        </labValue>
        <dataRef ind=”normSpecific” propertySetId=”ps5” />
      </valueSet>
      <valueSet id=”vs6”>
        <labValue>
          <val unit=”point”>22</val>
          <type>rational</type>
        </labValue>
        <dataRef ind=”normSpecific” propertySetId=”ps6” />
      </valueSet>
    </property>
    <propertySet id=”ps1”>
      <valueSetRelations>
        <ref valueSetId=”vs1” />
      </valueSetRelations>
      <dataRef>
        <ref begin=”0” end=”2” id=”nd1” />
      </dataRef>
    </propertySet>
    <propertySet id=”ps2”>
      <valueSetRelations>
        <ref valueSetId=”vs2” />
      </valueSetRelations>
      <dataRef>
```

```
        <ref begin=”3” end=”6” id=”nd1” />
      </dataRef>
    </propertySet>
    <propertySet id=”ps3”>
      <valueSetRelations>
        <ref valueSetId=”vs3” />
      </valueSetRelations>
      <dataRef>
        <ref begin=”7” end=”9” id=”nd1” />
      </dataRef>
    </propertySet>
    <propertySet id=”ps4”>
      <valueSetRelations>
        <ref valueSetId=”vs4” />
      </valueSetRelations>
      <dataRef>
        <ref begin=”10” end=”13” id=”nd1” />
      </dataRef>
    </propertySet>
    <propertySet id=”ps5”>
      <valueSetRelations>
        <ref valueSetId=”vs5” />
      </valueSetRelations>
      <dataRef>
        <ref begin=”14” end=”17” id=”nd1” />
      </dataRef>
    </propertySet>
    <propertySet id=”ps6”>
      <valueSetRelations>
        <ref valueSetId=”vs6” />
      </valueSetRelations>
      <dataRef>
        <ref begin=”18” end=”22” id=”nd1” />
      </dataRef>
    </propertySet>
  </object>
</xcdl>
```

As you can see, every `valueSet` represents one occurrence of the abstract property. The description of this occurrence is given by the `labValue`-Node, in which the two XML-nodes `val` and `type` with their attributes and values store the values which differentiate this `valueSet` from the others of the same basic property. Every `valueSet` is bound to a `propertySet`, every `propertySet` links to a short sequence inside the normData-Node. With this construct, it is much more simple for a program like the "comparator" not only to handle the occurrences of different properties but also of their values.

The `labValue`-node with its XML-nodes `val` and `type` has to be explained: The `value`-node describes a simple value. Different file formats use many different units. Because of this, we have decided not to try to map all units onto one "basic-unit", but to store the unit itself in an attribute called '`unit`'. This attribute is an enumeration restricted to `bit`, `twip`, `pixel`, `inch`, `meter`, `palette` and `point` an enumeration which presumably will have to be augmented in the future.

The XML-node '`type`' with its text-node describes the base type, upon which a program can base its selection of the metrics to do statistical calculations with this property or valueSet. In our example, the text-node is set to `rational`. For the "comparator" this means, that it can use metrics giving more precise results, than a simple 'true' or 'false' for the comparison of this property between two XCDL documents. But the XML-node '`type`' is also the place to specify other basic characteristics of the normData. Its group-attribute is used to tell programs, that the normData-section to which it is related has to be interpreted in groups of bytes with the length specified by this attribute. For example, if you read pixel-data from an image, in some cases one pixel is described by three colour-values, in other cases by four colour-values. The group-attribute tells the "comparator", how many normData-bytes are to be used for one basic unit, one pixel in this case.

If you remember object-references and the usage of properties and propertySets, it should now be clear that and how objects can also be used as properties.

Property (XML-Node)

The XML-node 'property' is a container where information can be accumulated, which can describe normData, an object or the whole XCDL. The informations are saved in other XML-nodes within the container. The property itself is not very complex.

This is a child of the node:

`object`

Attributes:

`id` - This is the mandatory and unique identifier for a property inside an object. It is recommended to make `ids` unique within the whole XCDL document.

`source` - This is an optional attribute indicating where the property has been derived from. Possible values are `raw`, `implicit` and `added`. In most cases, the value `raw` is set to show the property is read from the file directly. But if you think about the comparison of two files from different format specification, it is possible, that the same information is given in two files, but one file contains an explicit specification of a property, while the other does not so, as the specification of its format says that *all* files of this format share this property. In these cases, where properties are implicitly set by the format specification of the source file, the attribute 'source' is `implicit`. If the property is given neither by the format specification nor the file itself, but derived during processing, the attribute source is set to '`added`'. An example for this is the file size, or reconstructed data.

`cat` - This optional attribute categorizes the property. For most cases, `cat` is set to `descr`. The property describes a part of the normData or the object. If `cat` is set to `hist`, historical data are stored in the property, for example a special compression algorithm used in this context. The value `cont` for the attribute `cat` stands for relating the property to a byte sequence directly. The last possible value is `extern` for holding information about special hardware, software or other environmental settings, in which the original file was created.

Name (XML-Node)

This node within the XML-node name gives the property a mnemonic name, for example fontSize or italic. If you want to describe the name more explicit and in more than one shape, you have to use a valueSet.

This is a child of the node:

`property`

Attributes:

`id` - This mandatory attribute is NOT an identifier for the occurrence of the XML-node name, but for the name itself. It is currently defined within the appropriate namesLib and will be derived from the XCDL[1] ontology in the future.

`alias` - This optional attribute was created for storing information about the name of the property in the format specification of the source file. If you think about "image-width" and "image-length", these would be mapped to the same property in the XCDL document, might have been different within a source format, however. You are able to store this information for greater transparency of the process.

valueSet (XML-Node)

The XML-node valueSet is a container for more information about occurrences of a property.

This is a child of the node:

`property`

Attributes:

`id` - The `id` is a mandatory and unique identifier of this `valueSet`. Because this `id` is used outside the property, it has to be unique within the whole object.

labValue(XML-Node)

The XML-node labValue is a container for a pair consisting of one XML-node `val` and one XML-node `type`.

This is a child of the node:

`valueSet`

[1] See Planets deliverable PC/2 – D9

Attributes

none

val (XML-Node)

The XML-node `val` stores information about the value of an occurring property. Because we do not want to map different units of measurement to one basic one, the attribute 'unit' specifies which one has been used in the original file, allowing correct use of the numeric values within the comparison-metrics. The element <val> has a numeric or string value.

This is a child of the node:

`labValue`

Attributes:

`unit` - With the attribute `unit` you will be able to bind a unit to the value of the element <val>. This attribute is optional. You can choose between `bit`, `twip`, `pixel`, `inch`, `meter`, `palette` and `point`.

Type (XML-Node)

The XML-node `type` gives you information about the type of the variable, used in the corresponding XML-node `val`. Like the `unit`-attribute in the XML-node `val`, for better possibilities of interpretation and selecting the appropriate metrics in comparisons of different occurrences of that property in different files..

This is a child of the node:

`labValue`

Attributes:

`group` - This optional attribute stores the information about how many bytes of the linked normData describe one basic unit of these data. It expects an integer value. If it is not set, the value `1` is assumed.

Allowed values:

`int`, `rational`, `string`, `XCLLabel` for enumerations, `bool` and `time` as defined in ISO 8601.

rawValue (XML-Node)

Usage of this XML-node is optional. It is not illustrated and not shown in the property example above. The `rawValue` of a `valueSet` is the representation of the underlying "original" data, as read from the file. The only allowed change against what has been read from the file are conversions from the encoding of the file to UTF-16 for character data and to hexadecimal encoding for binary data. The binary data are stored as value of this XML element.

This is a child of the node:

`labValue`

Attributes:

`none`

dataRef (XML-Node)

With the XML-node `dataRef`, the whole `valueSet` that contains it is bound to a `propertySet,` as described in detail below. By default, the `valueSet` links to all normData in the same object.

This is a child of the node:

`valueSet`

Attributes:

`ind` - There are three valid values for the required attribute `ind`: `none, normSpecific` and `global`. With `normSpecific`, it is necessary to use the attribute `propertySetId`. Later, inside the linked `propertySet`, this property is applied to a specific sequence of the normData. If the attribute is `global`, this occurrence of the property is applied to all normData within this object. If the attribute is none, the valueSet is used without a relationship to any normData.

propertySetId - This attribute is required, if the attribute `ind` is set to normSpecific. It stores the `id` of the `properySet` it is linked to.

PropertySets

If you take a closer look at text-formats, you will find different possibilities to bind properties to the normData. Every text-attribute can be applied separately of all others to a text. This is important when different text-attributes overlap each other.

This is a **new *short*** *sentence.*

As you can see, the words `new short` are written bold, the words `short sentence` are italic. This can be expressed in XCDL in different ways. In the following, `propertySets` will be used to map `valueSets` to the normData:

```
<?xml version='1.0' encoding='utf-8'?>

<xcdl xmlns:xsi="http://www.planets-project.eu/xcl/schemas/xcl"
xsi:schemaLocation="http://www.planets-
project.eu/xcl/schemas/xcl/XCDLCore.xsd" id="0">

  <object id="o1">

    <normData id="nd1" type="text">This is a new short
sentence.</normData>

    <property id="p1">
```

```
      <name id="id161">bold</name>
      <valueSet id="vs1">
        <labValue>
          <val>default</val>
          <type>XCLLabel</type>
        </labValue>
        <dataRef ind=”normSpecific” propertySetId=”ps1” />
      </valueSet>
    </property>
    <property id="p2">
      <name id="id162">italic</name>
      <valueSet id="vs2">
        <labValue>
          <val>default</val>
          <type>XCLLabel</type>
        </labValue>
        <dataRef ind=”normSpecific” propertySetId=”ps2” />
      </valueSet>
    </property>
    <propertySet id="ps1">
      <valueSetRelations>
        <ref valueSetId=”vs1” />
      </valueSetRelations>
      <dataRef>
        <ref begin=”10” end=”18” id=”nd1” />
      </dataRef>
    </propertySet>
    <propertySet id="ps2">
      <valueSetRelations>
        <ref valueSetId=”vs2” />
      </valueSetRelations>
      <dataRef>
        <ref begin=”14” end=”27” id=”nd1” />
      </dataRef>
    </propertySet>
  </object>
</xcdl>
```

But it is also possible to understand propertySets as borders and containers for normData-areas with different combinations of properties.

```
<?xml version='1.0' encoding='utf-8'?>
```

```
<xcdl xmlns:xsi="http://www.planets-project.eu/xcl/schemas/xcl"
xsi:schemaLocation="http://www.planets-
project.eu/xcl/schemas/xcl/XCDLCore.xsd" id="0">
  <object id="o1">
    <normData id="nd1" type="text">This is a new short
sentence.</normData>
    <property id="p1">
      <name id="id161">bold</name>
      <valueSet id="vs1">
        <labValue>
          <val>default</val>
          <type>XCLLabel</type>
        </labValue>
        <dataRef ind=”normSpecific” propertySetId=”ps1” />
        <dataRef ind=”normSpecific” propertySetId=”ps2” />
      </valueSet>
    </property>
    <property id="p2">
      <name id="id162">italic</name>
      <valueSet id="vs2">
        <labValue>
          <val>default</val>
          <type>XCLLabel</type>
        </labValue>
        <dataRef ind=”normSpecific” propertySetId=”ps2” />
        <dataRef ind=”normSpecific” propertySetId=”ps3” />
      </valueSet>
    </property>
    <propertySet id="ps1">
      <valueSetRelations>
        <ref valueSetId=”vs1” />
      </valueSetRelations>
      <dataRef>
        <ref begin=”10” end=”12” id=”nd1” />
      </dataRef>
    </propertySet>
    <propertySet id="ps2">
      <valueSetRelations>
        <ref valueSetId=”vs1” />
        <ref valueSetId=”vs2” />
      </valueSetRelations>
      <dataRef>
```

```
        <ref begin=”14” end=”18” id=”nd1” />
      </dataRef>
    </propertySet>
    <propertySet id="ps3">
      <valueSetRelations>
        <ref valueSetId=”vs2” />
      </valueSetRelations>
      <dataRef>
        <ref begin=”20” end=”27” id=”nd1” />
      </dataRef>
    </propertySet>
  </object>
</xcdl>
```

Both XCDL-examples describe the same content of a file. There are many more ways to express the same sentence. For example, it is possible to create an own `valueSet` for each word and each property.

```
<?xml version='1.0' encoding='utf-8'?>
<xcdl xmlns:xsi="http://www.planets-project.eu/xcl/schemas/xcl"
xsi:schemaLocation="http://www.planets-
project.eu/xcl/schemas/xcl/XCDLCore.xsd" id="0">
  <object id="o1">
    <normData id="nd1" type="text">This is a new short
sentence.</normData>
    <property id="p1">
      <name id="id161">bold</name>
      <valueSet id="vs1">
        <labValue>
          <val>default</val>
          <type>XCLLabel</type>
        </labValue>
        <dataRef ind=”normSpecific” propertySetId=”ps1” />
      </valueSet>
      <valueSet id="vs3">
        <labValue>
          <val>default</val>
          <type>XCLLabel</type>
        </labValue>
        <dataRef ind=”normSpecific” propertySetId=”ps2” />
      </valueSet>
    </property>
    <property id="p2">
      <name id="id162">italic</name>
      <valueSet id="vs2">
        <labValue>
          <val>default</val>
          <type>XCLLabel</type>
        </labValue>
        <dataRef ind=”normSpecific” propertySetId=”ps2” />
        <dataRef ind=”normSpecific” propertySetId=”ps3” />
      </valueSet>
    </property>
    <propertySet id="ps1">
      <valueSetRelations>
        <ref valueSetId=”vs1” />
      </valueSetRelations>
      <dataRef>
        <ref begin=”10” end=”12” id=”nd1” />
      </dataRef>
    </propertySet>
    <propertySet id="ps2">
      <valueSetRelations>
        <ref valueSetId=”vs3” />
        <ref valueSetId=”vs2” />
      </valueSetRelations>
      <dataRef>
        <ref begin=”14” end=”18” id=”nd1” />
      </dataRef>
    </propertySet>
    <propertySet id="ps3">
      <valueSetRelations>
        <ref valueSetId=”vs2” />
      </valueSetRelations>
      <dataRef>
        <ref begin=”20” end=”27” id=”nd1” />
      </dataRef>
    </propertySet>
  </object>
</xcdl>
```

You may have seen that there are two `valueSets` of the property 'bold'. This is not the best way to represent this, but absolutely valid according to the XCDL specification. And it is also possible to construct an own `propertySet` for each `valueSet`.

```
<?xml version='1.0' encoding='utf-8'?>
<xcdl xmlns:xsi="http://www.planets-project.eu/xcl/schemas/xcl"
xsi:schemaLocation="http://www.planets-
project.eu/xcl/schemas/xcl/XCDLCore.xsd" id="0">
  <object id="o1">
    <normData id="nd1" type="text">This is a new short
sentence.</normData>
    <property id="p1">
      <name id="id161">bold</name>
      <valueSet id="vs1">
        <labValue>
          <val>default</val>
          <type>XCLLabel</type>
        </labValue>
        <dataRef ind=”normSpecific” propertySetId=”ps1” />
      </valueSet>
      <valueSet id="vs3">
        <labValue>
          <val>default</val>
          <type>XCLLabel</type>
        </labValue>
        <dataRef ind=”normSpecific” propertySetId=”ps2” />
      </valueSet>
    </property>
    <property id="p2">
      <name id="id162">italic</name>
      <valueSet id="vs2">
        <labValue>
          <val>default</val>
          <type>XCLLabel</type>
        </labValue>
        <dataRef ind=”normSpecific” propertySetId=”ps2” />
        <dataRef ind=”normSpecific” propertySetId=”ps4” />
      </valueSet>
    </property>
    <propertySet id="ps1">
      <valueSetRelations>
        <ref valueSetId=”vs1” />
      </valueSetRelations>
      <dataRef>
        <ref begin=”10” end=”12” id=”nd1” />
      </dataRef>
    </propertySet>
    <propertySet id="ps2">
      <valueSetRelations>
        <ref valueSetId=”vs3” />
      </valueSetRelations>
      <dataRef>
        <ref begin=”14” end=”18” id=”nd1” />
      </dataRef>
    </propertySet>
    <propertySet id="ps4">
      <valueSetRelations>
        <ref valueSetId=”vs2” />
      </valueSetRelations>
      <dataRef>
        <ref begin=”14” end=”18” id=”nd1” />
      </dataRef>
    </propertySet>
    <propertySet id="ps3">
      <valueSetRelations>
        <ref valueSetId=”vs2” />
      </valueSetRelations>
      <dataRef>
        <ref begin=”20” end=”27” id=”nd1” />
      </dataRef>
    </propertySet>
  </object>
</xcdl>
```

In the example above one `propertySet`, that contained a few valueSets, has been split into two propertySets. There is no clear logic, which of this usages and connections of propertySets and valueSets has to be used. All examples describe the same sentence and its properties but a "comparator" or rather its software engineer will find it difficult to process all of them.

It is very important, therefore, that we also define rules on how to select between syntactically legal constructions of the XCDL those, which should be used under normal circumstances for optimal processing. We propose that one property and its name can only exist once in an object. Same for valueSets: As a special case of a property it should only be defined once. If one valueSet is needed more than once, it should be bound several times but not be generated more than once.

If you read normData sequential, you tie a pool of properties via propertySets to the normData. If the constellation of the properties related to the normDatas changes now, a new propertySet should be used. PropertySets should not overlap, even though the XCDL specification allows it for reasons of future applications. But a propertySet can and also should be used repeatedly, when this particular constellation of properties is used again.

With the aid of these simple rules propertySets constructed from different formats will be quite similar. Even when other tools than the "extractor" are used to generate XCDLs, this rules are able to improve the structural similarity of the generated XCDL documents – and ease their usage by other software as a consequence.

propertySet (XML-Node)

The XML-node `propertySet` is a container for further XML-nodes required to combine properties into one logical unit. This unit can be linked more easily to normData than it would be to link each property separately. Furthermore, it is also easier to compare parts of normData for similarity property-relations. When we observe the rules explained in the last section, linking normData to the same propertySet implies the relation to the same set of properties for all of them. This is a child of the node:

`object`

Attributes:

`id` - Not only do XML-nodes inside propertySets link to XML-nodes outside, but also XML-nodes outside of them link back into the propertySets. Therefore propertySets have to have an `id`, which is unique within the whole object.

valueSetRelations (XML-Node)

As described previously, propertySets are bound to valueSets of the same object. Because one propertySet can be connected to more than one valueSet, the XML-node `valueSetRelations` is a container for storing all references to valueSets, as will be explained later.

This is a child of node:

`propertySet`

Attributes:

`none`

ref - inside valueSetRelations (XML-Node)

The XML-node `ref` inside the XML-container valueSetRelations describes a link to a valueSet inside the same object. Conceptually it also points to the propertySet within which it is contained. It has no explicit pointer to a propertySet, however. One XML-node ref represents exactly one relation from one propertySet to one valueSet.

This is a child of the node:

`valueSetRelations`

Attributes:

`valueSetId` - This attribute is the pointer to a valueSet in the same object. It stores the `id` of a valueSet. The attribute `valueSetId` is required.

dataRef (XML-Node)

As has been shown before, propertySets link to parts of the normData and represent a collection of properties. If these collections are used more than once, a propertySet should not be created anew, but used again. For this reason there is potentially more than one relation to normData-nodes. All references from a propertySet to normData are collected into the XML-container `dataRef`.

This is a child of the node:

`propertySet`

Attributes:

`none`

ref - inside dataRef (XML-Node)

In this context the XML-node `ref` always points to a part of normData or to the normData-structure as a whole. In the first case, two offsets are specified in the `ref`-node, using the two attributes begin and end. These offsets are defined by counting characters within the XML-node normData, starting with `0`. Of course, you have to specify the identifier of the normData-node to which this relationship shall be constructed as well.

This is a child of the node:

`dataRef`

Attributes:

`id` - This is the identifier of the normData-node to which, the propertySet containing this ref-node, is related to. This `id` is required.

`begin` - This is an offset from the first character of the normData, counting from `0`, to the first character using the related propertySet. This attribute is required, even if it is `0`.

`end` - This is an offset from the first character of the nromData, counting from `0`, to the last character using the related propertySet. This attribute is required, even if it is the last character of the normData-String.

3. Static XCDL Model

There have been several proposals from third parties, to use the XCDL for other purposes than a direct comparison of two source formats. We feel flattered by these indications of interest and confidence in the language. Within Planets we have to take care, however, that providing support to these ideas we do not damage the main purpose of these language within the project. So, while other usages of the language could and should be discussed, for the purposes of Planets we propose to proceed as follows.

An XCDL document can be constructed by Planets' "extractor" program, based upon a format-description expressed in XCEL to extract data from a file and generate an XCDL document. But this is not the only way to generate XCDLs. Other programs can, and are currently prepared to, extract data from files and save them as XCDL´documents. It is also possible to specify data directly in XCDL and use XCDL as a primary format in its own right.

Whereas the first possibility - extract data into a XCDL document to support more formats than the "extractor" can currently support - is certainly a useful approach, the use of XCDL as a primary format in its own right is currently not recommended by us for three reasons:

First, XCDL is currently supposed to act as a medium, with which to control the results of a migration, being independent of the sources as well as the target format of the migration. If an XCDL document is used directly to save data, the benefit of this “neutral” expression of the file content is lost. If an archive of images has been converted into XCDL documents, it is not possible to evaluate the success of the migration because there is no comparison format, with which we could check the result of this conversion.

The second reason against the use of XCDL itself as an archival format, are aspects of the design of XCDL itself. XCDL possesses no techniques for, for example, version controlling or unique file identification. Therefore XCDL would clearly be unsuitable as a primary archival format. Only if the XCDL documents themselves become connected to additional information, say using the METS standard, that situation *might* be mitigated.

And last but not least, the XCDL has not been designed for efficiency. During the XCDL creation progress all data are decompressed and saved uncompressed, organized in – in principle - human-readable units. This helps to compare the data but does not help to process them for most purposes. The enormous amount of data, which are generated by the application of the XCDL to images for example, makes XCDL clearly not a candidate for long term archiving within current repository capacities.

Therefore we believe that in its current form the XCDL should be used for comparison purposes only and not be updated, after the extraction process. Nevertheless it is possible to save additions to the information extracted originally. The object-model of the XCDL is able to bind in external XCDLs. If, for example, you try to save image annotations in XCDL and want to save them within the XCDL document describing the image, it is possible to generate the annotations as an XCDL-object and connect this XCDL object with the original data extracted from the image. Generally, however, we do not advice to modify XCDL documents, once they have been created.

XCDL documents *could* be used for archiving purposes and not only to control migration processes; this would need dedicated and not trivial work, therefore, which examines principles of file organisation and repository structures suitable for such an approach.

4. Final Words for now

During the last weeks and months the development of the XCDL has been guided by two principles. On the one hand we tried to keep the structure of XCDL as simple as possible, while supporting a reasonable comparison of two XCDL documents. This is only possible in an acceptable way if the structure of XCDLs is kept as flat as possible. On the other hand it is important that this structure is able to contain as much information as possible. It is not easy to develop a format, that is easy to understand and simultaneously able to save information from text files, picture formats and multimedia content.

If the XCDL is going to be used more generally and new constructs will ´have to be accommodated, both the specification of the XCDL and the recommendations for its usage will have to be augmented. We think though, that the current model is sufficiently general, that these modifications will be minor.

5. Modelling information

5.1 Limitations of the current XCL based approach

The XCDL has been designed to allow the modelling of digital content stored in different ways. As it is presented now, it has been derived from empirical and pragmatic approaches, where the kind of data contained in file formats which are most frequently discussed within the preservation community, provided many of the guiding principles for development. We consider this approach very useful; during the work on XCDL and its sister language XCEL, it became apparent, however, from discussions both within Planets as well as beyond the project, that there exist expectations for the treatment of characteristics of digital content, which go beyond some implicit limits of the empirical approach we have taken.

In one of the more recent documents of the W3C we read:

"The [XML 1.0] defines the XML language using a BNF grammar. Although a number of data models have been built on top of XML, as a syntactically defined language, XML is in itself data model agnostic. As stated earlier, an XML format is a format which is capable of representing the information in an XML document. Information, however, is in the eye of the beholder; what constitutes information, as opposed to just data, depends on the data model on which an XML processor is based."[2]

The statement is trivial from a computer science point of view: That "information" consists of data interpreted by some rules which are external to the data interpreted, is one of the most basic concepts of computer science. That the W3C considers it necessary to emphasize this very basic concept in a key paper shows, that it is not so present in some of the recent discussions about standardizing various ways of representing information, than one would wish. Within preservation discussions the fact, that this differentiation is not followed always very closely, is responsible for serious and costly misunderstandings, when we discuss what should be preserved about digital data. Many aspects of a digital document do *not* reside in the data stored, but in the rules for representing the information built into the software – in the case of textual documents as stored within text processing systems, almost all aspects of page layout are based upon parameters of the software displaying the text, not upon characteristics stored within a file.

Indeed, as has been discussed in detail in another (internal) deliverable of the Planets project[3], texts stored within text processing systems (like MS Word or Open Office) and page descriptive languages (like PDF), follow almost mutually exclusive ideals of what they can and what they cannot preserve, between invocations of the rendering software. PDFs take great care to preserve the layout of a text; they do not try to capture its structure. Office documents take some considerable interest in preserving the structure of a document, but they found it traditionally difficult to guarantee consistent layout, even between invocations of the identical rendering software on two machines with slightly different local settings. There is some convergence in this field: The concept of a "tagged" PDF augments a layout driven representation with structural characteristics and particularly the OOXML format includes a very large amount of layout information into what is basically a representation of the structure of a text. It is difficult not to assume, that there will not remain some limits to these attempts. One should not forget, that the early decades of the development of modern markup languages have been completely guided by a very strong position in that respect: One of the primary reasons, may be the single one most important reason, for the development of SGML was an assumption, that the way in which textual information was presented on a specific device was accidental, while irrelevant for most purposes – therefore a representation of the semantic structure of content of a document was what would be needed. Seeing the offspring of this development harnessed now into an attempt at making the *rendering* of content permanent, is at least astonishing.

[2] XML Binary Characterization Properties, W3C Working Group Note 31 March 2005, accessible as http://www.w3.org/TR/xbc-properties/.

[3] PC2, deliverable 6, "Interaction testing benchmark". Not yet publicly available at the time of this writing.

On the other hand, human readers are quite good at mixing these two levels: For a human reader the fact that something is a table or a footnote is immediately apparent from the way in which they are presented in layout. So, as has been shown in the deliverable quoted above, two representations of a text in file formats, which follow completely different principles, may still result in two renderings, which for the human reader look almost indistinguishable. If this capability of a human evaluator shall be translated into the kind of automated comparison of potentially many millions of files across migration events, which the XC languages have been designed to support, the data structures described by these languages have to be examined, whether they have sufficiently much expressive power to support additional software layers.

The XCDL has currently been developed to support the following scenario:

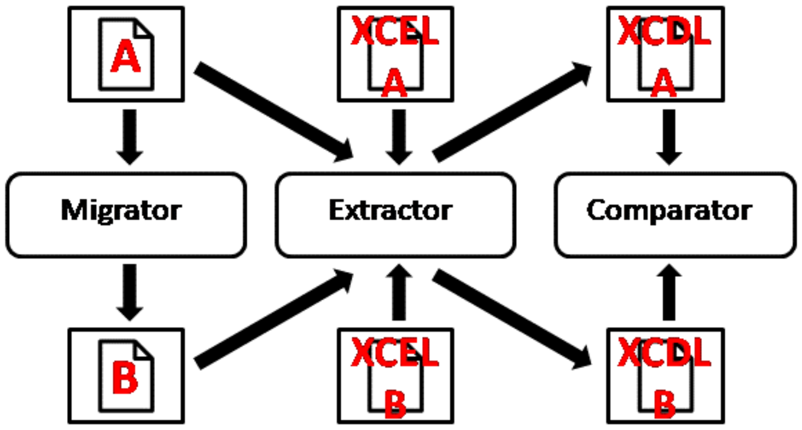

Figure 1: Basic XC Language Scenario

Or, to formulate this scenario from figure 1 once more: “To evaluate the result of a migration, we process the following procedure. (a) After a migration tool to be tested converts a file with the format ‘a’ into a file with the format ‘b’, a software module known as extractor reads formal specifications of these two formats, expressed in the language XCEL. Based on these specifications, it extracts the “characteristics” – or indeed the information contained within – those files and stores them represented by two documents encoded in the XCDL. A further software tool, which is called comparator, is able to compare these two documents and give an estimate of their similarity.”

This approach works with most image formats, as the information contained in an image file – the image itself and auxiliary information, like the owner of the copyright – are recognisable from the way in which the data are encoded, which are specified explicitly within the file format. It works, in other words, if “A” represents a TIF file and “B” a PNG file. A problem arises, though, when we try to process with the same model files, where some of the information is expressed by the human interpretation of a layout, say the statement “a centred line is a heading” or “sentences which follow at the bottom of a page after a raised number, which always starts a newline, and appears, also raised, in the preceding text, too, are footnotes”. Our model finds its limits, when “A” stands for a PDF file and “B” stands for a DOCX file.

As these interpretations do not reside in the file format specification, but in the mind of the reader, they cannot be extracted from the file based on the specification of the file format. This does not say, that it is impossible to derive, what in a text file is a heading or a footnote: The tools to extract these specifications from a file are based on a semantic analysis of the file’s layout, however. If we enter such a tool – an analyser – into the figure 1, we get figure 2, the scenario for an extended usage of the XC languages. Here we assume explicitly, that “A” stands for a file format, which encodes some of the information implicitly by layout – as in the PDF case – while “B” encodes the parts of the text explicitly – as in the DOCX case. The extended scenario assumes for this situation, that the representation of the text in the XCDL produced by the extractor software – labelled “XCDL A” in figure 2, can be used as input for a software system, which modifies the representation of the text in such a way, that in the derived representation – "XCDL A ' " – a header which has been recognised by its layout properties in "XCDL A" is represented in "XCDL A' " in exactly the same way, as this header is represented in "XCDL B", where it has been derived from a text processing type of document.

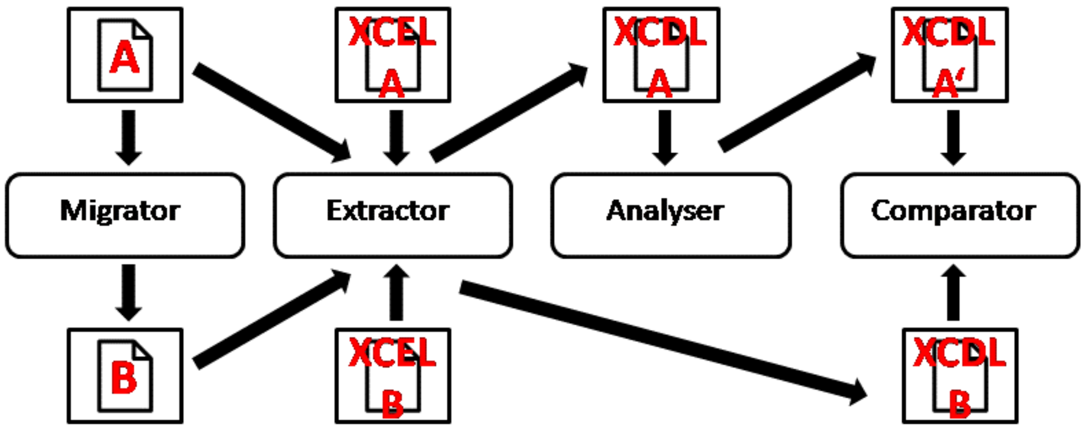

Figure 2: Extended XC Language Scenario

It should be pointed out, that as a model this has a very great advantage: As an analyser in this scenario is expected to produce output in the same representation it expects as input, it is obviously possible, to understand this as a recursive process, where repeated invocation of the analyser – or a set of analysers – "improves" the extracted information step by step, by recognizing more and more of the information expressed in the original file.

The pre-condition for such an extension, which we intend to pursue in the next stage of Planets, is, that we can represent with the XCDL all stages of such a process; that the XCDL is able, therefore, to represent the complete information contained within a file. The XCDL will have to be examined, however, how far (a) it uses an abstract model of information sufficiently powerful for that purpose, (b) how such a model can be formulated explicitly and (c) whether any changes to the XCDL become necessary as a result of such a model.

5.2 Towards an information model for preservation purposes

If we want to use the approach described above, we have to be able to compare files, assemblies of a number of files or different byte streams contained within a file, for the similarity of their content. To compare two things, they have to be reduced to a common abstract concept, which allows to concentrate on those characteristics, which are different between two instances of this concept: 'Comparing apples to oranges' is the popular expression of the impossibility to compare two objects, where the fundamental difference of the classes they represent is so pronounced, that the differences between the individual instances become impossible to evaluate. In a more formal way, we permanently operate with the notion that reducing representational irrelevancies to a core of formal, significant content allows us to compare seemingly different information. Nobody doubts that "MCMLXXXIV" and "1984" represent the same year, because the model "abstract number" is so deeply rooted in our mind, that we can reduce the representational rules of the two numeric systems used to their common internal representation within our minds without even noticing.

What we propose is, that: (1) The XCDL above uses – so far implicitly – an information model, that is able to describe data contained in files at a level of abstraction, where such comparisons are meaningful. (2) that model can be made explicit and, (3) if that is done sufficiently consistently, it allows to fine tune the language to support all imaginable needs for the extension of the original XCL scenario to the extended one which includes an additional step parsing information encoded by layout.

5.3 First sketch of an information model

In the XCDL specification introduced and discussed in the earlier chapters of this document, we have described a clear separation between two super classes of information contained within files: “normData”, which describe the data in the narrower sense contained in a file – the pixels of an image, the character sequence of a text – and “properties” describing how these data shall be processed generally (colour depth of an image) or handled in specific contexts (the font to be used for rendering a text). Properties, in that sense, can be applicable to a whole normData sequence (colour depth) or to a segment of it (font size applicable to characters n to m within a text).

We propose to generalize that preliminarily as follows:

Assumption 1:

Files contain or constitute information objects, which can be seen as instances of a relatively small number of base information classes: Texts, images, etc.

Assumption 2:

Each such information class consists of “content carrying tokens”.

Example: In the case of a text file, such content carrying tokens are typically characters encoded in the ASCII or Unicode. In the case of images, they are pixel values.

Assumption 3:

Each of the base classes configures such content carrying tokens in a specification configuration. Texts are basically *sequences* of tokens (usually called characters); images are *matrices* of tokens (usually called pixel information).

Assumption 4:

Each possible subsection of such a configuration of content carrying tokens can be described by arbitrarily many orthogonal properties, which describe how such tokens shall be processed.

Example: Processing here stands for purely mechanical as well as semantic operations. An abstract property can explain, how many bytes a content carrying token of an image has; it also can describe that characters n – m of a text are related to an entry in another file, where the geographical location associated with that substring (which presumably represents a topographical name) is administered.

Assumption 5:

This model is fully recursive. I.e., the character ‘n’ of a text may be described as one property among others by another textual object “xxx”, which represents, e.g., a footnote. This footnote in turn can be represented by a string of content carrying tokens – {‘x’, ‘x’, ‘x’} e.g. – each of which can be described by additional properties: font size, font name – or even another textual object “yyy” which connects that footnote to an entry in the index page of the primary textual object being represented.´

5.4 Relationship between Planets information model and XCDL ontology

The information model above operates on a conceptual level, although it has considerable practical implications. Basing the next generation of the XCL related software to its, assumes, that we need a very general but clear data structure on which we can build software, which successively extracts information from files, which is expressed in many different ways, up to and including the information which is expressed by layout in one file format, explicitly by specific properties encoded in a file format in another one. As such it is not a specific semantic, but a way to express the semantic content of a file, by dividing it into instances of base classes of information. (“A textfile containing an image.”) It can be said to provide a basic, abstract *syntax* for the representation of information, independent of its transitory encoding.

The XCDL *ontology,* on the other hand, provides a description of the relationships between the terms by which different parts of a file format are represented. It describes relationships between different terms used within different file formats to represent the same meaning. It can be said to provide an abstract *semantic* for the representation of information, independent of its transitory encoding.

6. Appendix

XML-Schema XCDLCore.xsd

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema  xmlns="http://www.planets-project.eu/xcl/schemas/xcl"
      xmlns:xs="http://www.w3.org/2001/XMLSchema"
      xmlns:nm="http://www.planets-project.eu/xcl/schemas/xcl"
      xmlns:xcdl="http://www.planets-project.eu/xcl/schemas/xcl"
      xmlns:xml="http://www.w3.org/XML/1998/namespace"
      targetNamespace="http://www.planets-project.eu/xcl/schemas/xcl"
      elementFormDefault="qualified"
      version="1.0"
      xml:lang="en" >

      <xs:import namespace="http://www.w3.org/XML/1998/namespace"
            schemaLocation="preserve.xsd"/>
      <xs:include schemaLocation="XCDLBasicTypes.xsd"/>
      <!--********************************** xcdl section  (root
element)********************************************* -->
      <xs:element name="xcdl">
            <xs:annotation>
                  <xs:documentation>
                        eXtensible Characterisation Description Language
(XCDL)
                  </xs:documentation>
            </xs:annotation>
            <xs:complexType>
                  <xs:complexContent>
                        <xs:extension base="xcdlType"/>
                  </xs:complexContent>
            </xs:complexType>
      </xs:element>
      <!-- ............complex type:  xcdlType ................. -->
      <xs:complexType name="xcdlType">
            <xs:annotation>
                  <xs:documentation>
                        A XCDL document describes digital objects. Every
xcdl description shall have an identification number.
                </xs:documentation>
            </xs:annotation>
            <xs:sequence>
                  <xs:element ref="object" maxOccurs="unbounded">
                        <xs:annotation>
                              <xs:documentation>
                                    An object is a string of content
carrying tokens (called normData) each token
                                    can be associated with different
meanings (called properties). Properties can
                                    either add an atomic meaning to the
referenced token or they can reference
                                    an other object with the token.
                              </xs:documentation>
                        </xs:annotation>
                  </xs:element>
            </xs:sequence>
            <xs:attribute name="id" type="xcdl:idType02Type"
use="required"/>
```

```
      </xs:complexType>
      <!-- *****************************  object section (child of:
'xcdl')  *********************************************** -->
      <xs:element name="object">
            <xs:annotation>
                  <xs:documentation>
                        Function: Wrapper element for objects to be
described through a xcdl.
                        Every object shall have an identification number.
                        The native format of the object may be added.
                  </xs:documentation>
            </xs:annotation>
            <xs:complexType>
                  <xs:sequence>
                        <xs:element ref="data" minOccurs="0"/>
                        <xs:element ref="normData" minOccurs="0"
maxOccurs="unbounded"/>
                        <xs:element ref="property" minOccurs="0"
maxOccurs="unbounded"/>
                        <xs:element ref="propertySet" minOccurs="0"
maxOccurs="unbounded">
                              <xs:annotation>
                                    <xs:documentation>
                                          A propertySet provides one
possibility to connect atomic properties to a
                                          more complex set of properties.
References to a specific content token are always
                                          expressed with the aim of a
propertySet.
                                          PropertySets have no option to
model the direction of a property relation. If you
                                          want to provide such a
direction you can use a new object instead where all properties
                                          are connected to the normData.
                                    </xs:documentation>
                              </xs:annotation>
                        </xs:element>
                  </xs:sequence>
                  <xs:attribute name="id" type="xcdl:idType01Type"
use="required"/>
            </xs:complexType>
      </xs:element>
      <!-- *****************************  propertySet section (child of:
'xcdl')  *********************************************** -->
      <xs:element name="propertySet">
            <xs:complexType>
                  <xs:annotation>
                        <xs:documentation>
                              A propertySet can reference different
valueSets of different properties to apply them
                              to a certain token of the normData.
                        </xs:documentation>
                  </xs:annotation>
                  <xs:sequence>
                        <xs:element name="valueSetRelations">
                              <xs:annotation>
                                    <xs:documentation>
                                          A ValueSetRelation can contain
multiple elements for referencing different valueSets
                                    </xs:documentation>
                              </xs:annotation>
                              <xs:complexType>
                                    <xs:sequence>
```

```
                                        <xs:element name="ref"
maxOccurs="unbounded">
                                              <xs:complexType>
                                                    <xs:attribute
name="valueSetId" type="xs:ID" use="required"/>
                                              </xs:complexType>
                                        </xs:element>
                                  </xs:sequence>
                            </xs:complexType>
                      </xs:element>
                      <xs:element name="dataRef" minOccurs="0">
                            <xs:complexType>
                                  <xs:annotation>
                                        <xs:documentation>
                                              A DataRef can be used to
apply the references from within the valueSetRelation element
                                              to a certain position
(token) of the normData String. For that it must store the begin and end
                                              of the token in form of
an index and the normData-ID to identify the correct normData.
                                        </xs:documentation>
                                  </xs:annotation>
                                  <xs:sequence>
                                        <xs:element name="ref"
maxOccurs="unbounded">
                                              <xs:complexType>
                                                    <xs:attribute
name="begin" type="xs:int" use="required"/>
                                                    <xs:attribute
name="end" type="xs:int" use="required"/>
                                                    <xs:attribute
name="id" type="xs:ID" use="required"/>
                                              </xs:complexType>
                                        </xs:element>
                                  </xs:sequence>
                            </xs:complexType>
                      </xs:element>
                </xs:sequence>
                <xs:attribute name="id" type="xs:ID"></xs:attribute>
          </xs:complexType>
    </xs:element>
    <!-- ***************************  data section (child of: 'object')
****************************************  -->
    <xs:element name="data">
          <xs:annotation>
                <xs:documentation>
                      Function: Wraps the full source objects data.
                      For relation and reference purposes, an
identification number is required.
                </xs:documentation>
          </xs:annotation>
          <xs:complexType>
                <xs:simpleContent>
                      <xs:extension base="xcdl:unionType01Type">
                            <xs:attribute name="id"
type="xcdl:idType01Type" use="required"/>
                      </xs:extension>
                </xs:simpleContent>
          </xs:complexType>
    </xs:element>
    <!-- *************************** normalized data section (child of:
'object')  ***************************** -->
    <xs:element name="normData">
```

```
            <xs:annotation>
                  <xs:documentation>
                        Function: Wraps normalized data.
                  </xs:documentation>
            </xs:annotation>
            <xs:complexType>
                  <xs:simpleContent>
                        <xs:extension base="xcdl:unionType01Type">
                              <xs:attribute name="id"
type="xcdl:idType01Type" use="required"/>
                              <xs:attribute name="type"
type="xcdl:informType" use="required"/>
                        </xs:extension>
                  </xs:simpleContent>
            </xs:complexType>
      </xs:element>
      <!-- ************************  Property section (child of:
'object')  ******************************* -->
      <xs:element name="property">
            <xs:annotation>
                  <xs:documentation>
                        Function: Wraps the objects properties.
                        A property shall have an identification number
within the xcdl description.
                  </xs:documentation>
            </xs:annotation>
            <xs:complexType>
                  <xs:sequence>
                        <xs:element ref="name"/>
                        <xs:element ref="valueSet"
maxOccurs="unbounded"/>
                  </xs:sequence>
                  <xs:attribute name="id" type="xcdl:idType01Type"
use="required"/>
                  <xs:attribute name="source" type="sourceType"
use="optional"/>
                  <xs:attribute name="cat" type="catType"
use="optional"/>
            </xs:complexType>
      </xs:element>
      <!--  ............simple type: sourceType............:  -->
      <xs:simpleType name="sourceType">
            <xs:annotation>
                  <xs:documentation>
                        the source the property is derived from.
                        'raw' =derived from the source object;
'implicit'=property is not fixed to the source object but derived from
the source objects format specification
                        'added'= property is not raw and implicit, but
derived from the file, e.g. filesize or original filename
                  </xs:documentation>
            </xs:annotation>
            <xs:restriction base="xs:token">
                  <xs:enumeration value="raw"/>
                  <xs:enumeration value="implicit"/>
                  <xs:enumeration value="added"/>
            </xs:restriction>
      </xs:simpleType>
      <!--  ............simple type: catType............:  -->
      <xs:simpleType name="catType">
            <xs:annotation>
                  <xs:documentation>
```

```
                        the properties category: 'descr'=descriptive
property, i.e. occurence of object describing property; 'hist'= history
property,
                        i.e. property that may appear in a different
shape in the source object which may be resolved in the xcdl description
(e.g., compressed data);
                        'cont'= content property, i.e. relating directly
to a byte sequence; 'extern'= property that refers to external item, i.e.
not related to objects data, e.g.,
                        software and hardware used to create the object.
                  </xs:documentation>
            </xs:annotation>
            <xs:restriction base="xs:token">
                  <xs:enumeration value="descr"/>
                  <xs:enumeration value="hist"/>
                  <xs:enumeration value="cont"/>
                  <xs:enumeration value="extern"/>
            </xs:restriction>
      </xs:simpleType>
      <!-- ************************  property name section (child of:
'property')  **************************************  -->
      <xs:element name="name">
            <xs:annotation>
                  <xs:documentation>
                        Function: Wraps a unique property name, defined
by a xcdl names library.
                        Namings for properties which refer to identical
issues may differ depending on the format.
                        The different term may be added using the 'alias'
attribute.
                  </xs:documentation>
            </xs:annotation>
            <xs:complexType>
                  <xs:simpleContent>
                        <xs:extension base="xcdl:nameType">
                              <xs:attribute name="id" type="xs:string"
use="required"/>
                              <xs:attribute name="alias" type="xs:string"
use="optional"/>
                        </xs:extension>
                  </xs:simpleContent>
            </xs:complexType>
      </xs:element>
      <!-- ************************  property value set section  (child
of: 'property') ********************************* -->
      <xs:element name="valueSet">
            <xs:annotation>
                  <xs:documentation>
                        Function: Wrapper element for the properties raw
and labelled values.
                        Every value set shall have an identification
number.
                  </xs:documentation>
            </xs:annotation>
            <xs:complexType>
                  <xs:sequence>
                        <xs:element ref="rawValue" minOccurs="0"/>
                        <xs:element ref="labValue" minOccurs="0"/>
                        <xs:element name="objectRef" minOccurs="0">
                              <xs:simpleType>
                                    <xs:restriction base="xs:string">
                                          <xs:pattern
value="(file://.*|\.):.*"></xs:pattern>
```

```
                                    </xs:restriction>
                              </xs:simpleType>
                        </xs:element>
                        <xs:element ref="dataRef" minOccurs="0"/>
                  </xs:sequence>
                  <xs:attribute name="id" type="xcdl:idType01Type"
use="required"/>
            </xs:complexType>
      </xs:element>
      <!-- ************************** raw value section (child of:
'valueSet') ************************************** -->
      <xs:element name="rawValue">
            <xs:annotation>
                  <xs:documentation>
                        Function: Wraps the distinct raw value, as
extracted from the source object ;
                        by default bytes shall be encoded in UTF-16 for
non-binary data, in hex numbers for binary data.
                  </xs:documentation>
            </xs:annotation>
            <xs:complexType>
                  <xs:simpleContent>
                        <xs:extension base="xcdl:unionType01Type"/>
                  </xs:simpleContent>
            </xs:complexType>
      </xs:element>
      <!--  ************************  labelled value section (child of:
'valueSet') ****************************** -->
      <xs:element name="labValue">
            <xs:annotation>
                  <xs:documentation>
                        Function: Wrapping element for labelled value.
                        A labelled value shall be expressed by its
distinct value and its type.
                  </xs:documentation>
            </xs:annotation>
            <xs:complexType>
                  <xs:sequence>
                        <xs:element name="val" maxOccurs="unbounded">
                              <xs:annotation>
                                    <xs:documentation>
                                          The distinct labelled value.
                                          This can either be a UTF-16
encoded string, an integer or decimal number or a fixed label defined in
simple type 'xcdlFixedLabls'.
                                          For an accurate representation
some properties values, especially those expressed in integers,
                                          may require additional
measurement information.
                                    </xs:documentation>
                              </xs:annotation>
                              <xs:complexType>
                                    <xs:simpleContent>
                                          <xs:extension
base="xcdl:unionType10Type">
                                                <xs:attribute name="unit"
type="xcdl:measureType" use="optional"/>
                                          </xs:extension>
                                    </xs:simpleContent>
                              </xs:complexType>
                        </xs:element>
                        <xs:element name="type" maxOccurs="unbounded">
                              <xs:annotation>
```

```
                                    <xs:documentation>
                                          Function: Wraps type of the
labelled value.
                                          Some properties distinct values
may be expressed as a sequence of values which are repeatable groups.
                                          E.g.: a palette with entries
for colours mixed from red, green and blue channels may be x times
repeatable
                                          depending on the bit depth. In
this case the 'group' attribute is set on value '3', telling the reading
tool that
                                          a meaningful unit consists of
triplets.
                                    </xs:documentation>
                              </xs:annotation>
                              <xs:complexType>
                                    <xs:simpleContent>
                                          <xs:extension
base="xcdl:labValType">
                                                <xs:attribute
name="group" type="xs:unsignedInt" use="optional"/>
                                          </xs:extension>
                                    </xs:simpleContent>
                              </xs:complexType>
                        </xs:element>
                  </xs:sequence>
            </xs:complexType>
      </xs:element>
      <!-- ****************** data reference section (child of:
'valueSet') **************************  -->
      <xs:element name="dataRef">
            <xs:annotation>
                  <xs:documentation>
                        Function: Reference to data.
                        This can either be the source data (element
'data')
                        or normalized data (element 'normData').
                        Attribute 'ind': Indicator for the type of
reference.
                  </xs:documentation>
            </xs:annotation>
            <xs:complexType>
                  <xs:attribute name="ind" type="dataRefType"
use="required"/>
                  <xs:attribute name="propertySetId" type="xs:ID"
use="optional"/>
            </xs:complexType>
      </xs:element>
      <!-- ........simple type: 'dataRefType': ....... -->
      <xs:simpleType name="dataRefType">
            <xs:annotation>
                  <xs:documentation>
                        Type of data reference: none= no reference to
data; normSpecific= reference to specific normalized data;
                        global= reference to all normalized data in the
same object. If value is 'normSpecific', 'ref' element shall be included.
                  </xs:documentation>
            </xs:annotation>
            <xs:restriction base="xs:token">
                  <xs:enumeration value="none"/>
                  <xs:enumeration value="global"/>
                  <xs:enumeration value="normSpecific"/>
            </xs:restriction>
      </xs:simpleType>
      <!-- ......... element 'ref':  .............. -->
      <xs:element name="ref">
            <xs:annotation>
                  <xs:documentation>
                        Function: distinct data coordinates for
references.
                        Startposition and Endposition of the bytes within
the
                        referenced data shall be declared using the
attributes.
                        Also required is an identification number for
each 'ref' element.
                  </xs:documentation>
            </xs:annotation>
            <xs:complexType>
                  <xs:attribute name="id" type="xcdl:idType01Type"
use="required"/>
                  <xs:attribute name="start" type="xs:unsignedLong"
use="required"/>
                  <xs:attribute name="end" type="xs:unsignedLong"
use="required"/>
            </xs:complexType>
      </xs:element>
</xs:schema>
```

XML-Schema XCDLBasicTypes.xsd

```
<?xml version="1.0" encoding="UTF-8"?>
      <xs:schema
      xmlns="http://www.planets-project.eu/xcl/schemas/xcl"
      xmlns:xs="http://www.w3.org/2001/XMLSchema"
      xmlns:nm="http://www.planets-project.eu/xcl/schemas/xcl"
      xmlns:xcdl="http://www.planets-project.eu/xcl/schemas/xcl"
      targetNamespace="http://www.planets-project.eu/xcl/schemas/xcl"
      elementFormDefault="qualified"
      version="1.0"
      xml:lang="en">
      <xs:include schemaLocation="xcel/XCLBasicNamesLib.xsd" />
      <!-- ***  union types  *** -->
      <xs:simpleType name="unionType01Type">
            <xs:union memberTypes="xs:string xs:hexBinary"/>
      </xs:simpleType>
      <xs:simpleType name="unionType02Type">
            <xs:union memberTypes="xs:string xs:integer"/>
      </xs:simpleType>
      <xs:simpleType name="unionType10Type">
            <xs:union memberTypes="xs:string xs:decimal
xcdl:fixLabelsType"/>
      </xs:simpleType>
<!-- ***  identification number types  *** -->
      <xs:simpleType name="idType01Type">
            <xs:restriction base="xs:string"/>
      </xs:simpleType>
      <xs:simpleType name="idType02Type">
            <xs:restriction base="xs:string"/>
      </xs:simpleType>
      <!-- ***  information types  *** -->
      <xs:simpleType name="informType">
```

```
            <xs:annotation>
                  <xs:documentation>
                        type of information represented by data.
                  </xs:documentation>
            </xs:annotation>
            <xs:restriction base="xs:string">
                  <xs:enumeration value="text"/>
                  <xs:enumeration value="image"/>
                  <xs:enumeration value="audio"/>
                  <xs:enumeration value="object"/>
                  <xs:enumeration value="other"/>
            </xs:restriction>
      </xs:simpleType>
      <!-- *** format identifier type ***  -->
      <xs:simpleType name="formatIdentifierType">
            <xs:restriction base="xs:string"/>
      </xs:simpleType>
      <!-- ***  measure types  *** -->
      <xs:simpleType name="measureType">
            <xs:restriction base="xs:string">
                  <xs:enumeration value="bit"/>
                  <xs:enumeration value="twip"/>
                  <xs:enumeration value="pixel"/>
                  <xs:enumeration value="inch"/>
                  <xs:enumeration value="meter"/>
                  <xs:enumeration value="palette"/>
                  <xs:enumeration value="point"/>
            </xs:restriction>
      </xs:simpleType>
      <!-- *** simple type for xcl defined namings for properties: *** --
>
      <xs:simpleType name="nameType">
      <xs:annotation>
                  <xs:documentation> union of xcl defined namings for
xcdl properties</xs:documentation>
            </xs:annotation>
            <xs:union memberTypes="nm:xclBasicNameDefinitions
xs:string"/>
      </xs:simpleType>
      <!-- *** simple type for xcl defined data types for xcdl labelled
value types: *** -->
      <xs:simpleType name="labValType">
      <xs:annotation>
                  <xs:documentation>derived from xcl defined data
types</xs:documentation>
            </xs:annotation>
            <xs:union memberTypes="nm:xclBasicNameDefinitions
xs:string"/>
      </xs:simpleType>
      <!-- *** simple type for xcl defined namings for xcdl properties'
labelled values: *** -->
      <xs:simpleType name="fixLabelsType">
      <xs:annotation>
                  <xs:documentation> union of xcl defined namings for
xcdl fixed labellings</xs:documentation>
            </xs:annotation>
            <xs:union memberTypes="nm:xclBasicNameDefinitions xs:string"
/>
      </xs:simpleType>
            <xs:simpleType name="spaceTypes">
                  <xs:restriction base="xs:token">
                        <xs:enumeration value="preserved"/>
                        <xs:enumeration value="default"/>
                  </xs:restriction>
            </xs:simpleType>
</xs:schema>
```

7. Reference

1. Jan Schnasse, Volker Heydegger et. al., XCEL Specification, 2008;

http://planetarium.hki.uni-koeln.de/public/XCL/xcl/XCLDocumentation/xcelDocu.htm

2. Metadata Encoding & Transmission Standard

    http://www.loc.gov/standards/mets/

3. Ecma International approves Office Open XML standard

    http://www.ecma-international.org/news/PressReleases/PR_TC45_Dec2006.htm

4. Volker Heydegger, Jan Schnasse et. Al, XCDL Specification (2006)