Project Number	IST-2006-033789
Project Title	PLANETS
Title of Deliverable	Final XCEL Specification
Deliverable Number	D8
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	4th July 2008
Author(s)	HKI – University of Cologne

Abstract
The aim of this document is to describe the eXtensible Characterisation Extraction Language (XCEL). It does so by providing a systematic description of the language elements and their usage, augmented by detailed examples of their application. Finally, the language is described formally by the inclusion of the XML schemas which define it.

Keyword list
XCEL, XCL, XCDL, Preservation Characteristics, Extensible Characterisation Extraction Language, Format Definition Language

Contributors

Person	Role	Partner	Contribution
Jan Schnasse	Author	UzK	
Volker Heydegger	Contributor	UzK	Common coaction
Elona Chudobkaite	Contributor	UzK	Common coaction

Document Approval

Person	Role	Partner
Adrian Brown	Sub-project lead	TNA
Hannes Kulovits	Reviewer	TUWIEN

Distribution

Person	Role	Partner

Revision History

Issue	Author	Date	Description
1.0	Jan Schnasse	31st May 2008	Original reference documentation

References

Ref.	Document	Date	Details and Version

EXECUTIVE SUMMARY

The specification provides a general overview about the Extensible Characterisation Extraction Language (XCEL) as well as a normative description of the language features. The document does also include several tutorials and code examples about the usage of the XCEL. Within the document it is shown that the XCEL provides all basic tools to describe a variety of specific file formats as well as more general file structures common to many of them.

Table of Contents

1. Introduction

1.1 About this Document

This document is the current specification of the eXtensible Characterisation Extraction Language (XCEL), as finalized on May 31st 2008 at the University at Cologne. It describes the XCEL 1.0b version and was reviewed by PLANETS members.
The specification consists of a normative specification of all language elements [XCEL Elements], informative parts [Overview, Extending the XCEL, XCEL Processor Rules], implementation details [XCEL Schema architectureXCEL Schema architecture] and examples [HowTo XCEL]. In the Appendix we provide the definition of the XCEL syntax available as a set of XML Schemas [see XCEL Schemas].
The XCEL 1.0b specification is strongly connected to specification of the eXtensible Characterisation Description Language (XCDL) finalised on May 31st 2008 as well. The specifications of both sub languages together are referred to as the eXtensible Characterisation Languages specification.

1.2 Overview

The eXtensible Characterisation Extraction Language (XCEL) is a file format description language for describing file structures to allow their parsing by generalized software. The main goal of the XCEL is therefore to provide all tools necessary for describing real-life file formats like PNG, TIFF, PDF or DOCX.[1] The XCEL is a declarative, descriptive, XML-based language that provides well defined mechanisms for extending certain parts of the language.
As an Extraction Language the XCEL has some similarities to other domain specific languages for describing file formats.[2] The Data Format Description Language (DFDL) has a number of common properties with the XCEL, there are significant differences, however. DFDL focuses on scientific data[3], XCEL is primarily targeted at file formats as typically held in libraries and archives. The DFDL is implemented as an extension of XML-Schema, the XCEL is a completely new language the syntax of which can be described with the XML-Schema language. Other approaches like PADS are focusing the processing of simple but large scale data formats.[4] The distinct characteristic of XCEL, however, is, that it does not extract purely technical entities – “a 3 x 256 array of one byte numbers” - but characteristics with a semantic meaning - “a colour lookup table”.
The usage of the XCEL is not limited to a specific field of work, nevertheless the language has been developed as part of an evaluation framework for the automated control of format conversions. Within this framework the result of applying an XCEL document to a file is always an XML-document with a structure that is defined in the XCDL specification. We strongly emphasize the independence of the different components – XCEL, extracting software and XCDL – but, nevertheless, the individual components have been designed to be able to work together smoothly. The following figure outlines the main use case of the evaluation framework.

[1] The PNG specification is available at <http://www.w3.org/TR/PNG/>
The TIFF specification is available at <http://partners.adobe.com/public/developer/tiff/index.html>
The PDF specification is available at <http://www.adobe.com/devnet/pdf/pdf_reference.html>
The DOCX specification is available at <http://www.ecma-international.org/news/TC45_current_work/TC45_available_docs.htm>
[2] K. Fisher, Y. Mandelbaum, D. Walker: The next 700 Data Description Languages,in: Conference record of the 33rd ACM SIGPLAN-SIGACT symposium on Principles of programming languages, p2-15, 2006. The text provides a formal overview about the scope of format description languages. The XCEL seems to cover all the structures that are described in the formal definitions mentioned in this paper.
[3] The DFDL is described by several papers on sourceforge: http://forge.gridforum.org/sf/projects/dfdl-wg
[4] K.Fisher, R. Gruber: PADS: A Domain-Specific Language for Processing Ad Hoc Data, in: Proceedings of the 2005 ACM SIGPLAN conference on Programming language design and implementation, p. 295-304, 2005. The paper describes a language based on C-Structures. One main design goal of PADS is a performant processing of large, simple structured files.

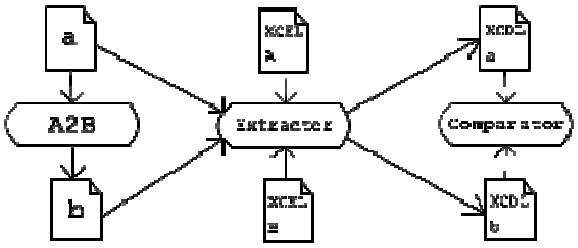

Figure 1: Evaluation of a format conversion: Conversion of an instance a of format A into an instance b of format B. Extraction of a with an XCEL for format A . Extraction of b with an XCEL for format B. Comparison of an XCDL instance of a to an XCDL instance of b.

The single components in this use case can be described as follows. 'A2B' refers to an arbitrary conversion tool that is used to convert a file into another one stored in a different format. 'XCEL A' and 'XCEL B' are **XCEL Document**s for both, the source and the target format. The Extractor is an **XCEL Processor** that is able to extract the content of a file using a given **XCEL Document** for the underlying format. The Extractor is able to translate the extracted content into a normalised XCDL representation. The figure shows that the Extractor is used two times to translate both files (a and b) to an XCDL representation. The Comparator is a tool that understands the XCDL format and is able to compare single components of the XCDL with different metrics.

Together all the components described can be used to estimate the degree of success of a format conversion, by measuring the degree with which the characteristics of the object stored in the first file format have been retained during the migration.
XCEL provides all tools needed to create formalized, machine interpretable descriptions of a file format. Beside their primary use as an extraction specification, **XCEL Document**s can also be seen as a meta-format to store the knowledge about a certain file format in machine-readable form. The XCEL allows to describe just parts of a file format. This can significantly ease the process of evaluation, e.g. if you already know that a certain conversion tool sometimes produces unexpected results on files with certain features, you can concentrate on describing those features in XCEL without thinking about the unproblematic parts of the format.

PNG 16Bit greyscale image

PNG to TIFF conversion result with photoshop

Figure 2: The figure shows a failed conversion of png into tiff using photoshop. This known error occurs only on 16Bit greyscale images. With the XCEL you can easily extract just the header information to find all files being at risk.

1.2.1 The XCEL Datamodel

XCEL uses a tree-like model to describe file structures. Each branch of the tree describes a certain part of the file format. It is the responsibility of the **XCEL Processor** to find the branches that provide a proper description of the file currently processed. If a branch describes a part of the file properly, we say 'the branch matches'. An **XCEL Tree** is a proper description of a certain file if the root element matches.
An **XCEL Tree** is made of **XCEL Element**s. The Item elements are container elements that build the branches of the tree. Depending on their type, Items can have different functions. Used for semantic structures they define logical groups. But usually Items are used to define that a certain sequence of elements must appear at a well defined position within the file [see Items].

Symbols are the second type of elements. They must appear as leaves in the tree. Usually Symbols are used to consume a piece of the byte stream and add information to it. Symbols can also be used to describe that a special pattern of data *must* occur. This allows Symbols to act as

conditions for the selection of matching branches in the tree [see HowTo express Conditions and Symbols].
The last type of element is called Processing. With Processings the XCEL Tree can be modified at runtime, e.g. By decompressing a byte stream which h as been stored in the file format according to a specific compression scheme. Processings can also configure the behaviour of the parser, e.g. skip white space, read byte-wise etc. [see Processings]

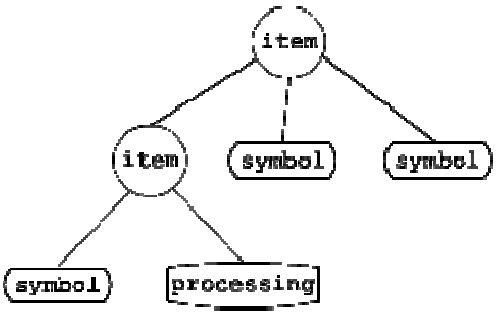

Figure 3: XCEL Tree

In principle the XCEL Tree describes the file sequentially from its beginning. A sequence is defined by the XCEL Elements it can contain and the order in which the elements can appear.
XCEL provides three different Content Models [see **Content Model**] for the definition of orders: *sequence*, *choice* and *all*. For non-sequential orders, XCEL Elements can be defined to start at a certain offset within the file. Offsets can be set dynamically at runtime, and XCEL provides a wide variety of methods to handle different kinds of offset definitions [see HowTo do Addressing].
The result of the XCEL-driven parsing of a file is stored in another tree. This **Result Tree** contains all XCEL Elements that have matched and associates the concrete values of the characteristics parsed from the file with the elements representing the abstract properties. The Result Tree is ordered according to the occurrence of matching elements. To provide a flexible interface there are additionally several indices and methods that allow to walk and query the tree by different criteria.

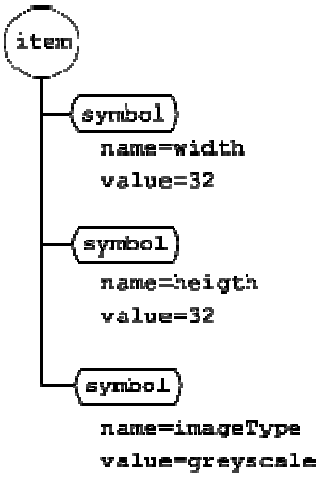

Figure 4: A Result Tree

1.2.2 The XCEL Main Structure

"Writing an XCEL" program consists in writing an XML document, which is valid against the schema structure described in the "XML schema architecture" described later in that document. While these schemas become very intricate, when covering the properties of present in complex file formats, the XCEL uses a very simple overall structure.

A complete **XCEL Document** is divided into four distinct parts. The three main parts are called 'preProcessing', 'formatDescription' and 'postProcessing'. All of these three parts must contain exactly one or zero **XCEL Tree**s. A fourth part is called 'templates'.

```
<xcdlDocument>
<preProcessing> ... </preProcessing>
      <formatDescription>  ... </formatDescription>
      <templates> ... </templates>
      <postProcessing> ... </postProcessing>
</xcdlDocument>
```

Within the templates section many XCEL Trees can appear. The idea of the templates section is to provide a possibility for defining recurring structures. Such structures can be reused in other sections of the XCEL.
The 'preProcessing' section was mainly introduced to provide a place to describe preliminary tasks. A good example is provided by formats which store their component files within a zip-container and thus need to be decompressed before parsing the content.
The 'formatDescription' section is the main part of the XCEL. The 'formatDescription' has two objectives: Firstly, to represent the structure of the format and secondly to relate the parts of the data that are expected to be extracted as characteristics of the file to the proper names of the properties of which they are instances.
The 'postProcessing' section is reserved for all actions that require the parsing of the file is completed. Examples for such actions are the application of filtering methods or normalisation tasks which can only be performed after all the parameters describing the method to be applied, as well as the data to which it is be applied have been retrieved.

1.3 Introductory Example

To conclude the overview this section introduces a simple example to give the reader a feeling about how the XCEL works.
Assuming we want to extract all surnames from a file that stores person names in a well defined format. The format definition reads as following:

„Every name is stored as a forename, followed by a single white space, followed by the surname, terminated by a carriage return. All names have variable length and are restricted to contain only ASCII characters.“

An example of this format might look like this:

```
Gottfried Leibniz
Alexander Baumgarten
Immanuel Kant
Friedrich Schiller
Johann Fichte
Friedrich Schelling
Georg Hegel
```

How can we describe this simple structure with an **XCEL Tree**? The first step is to create a container **Item** that describes the whole file.

```
<item xsi:type=“structuringItem“ identifier=“ourFile“ order=“sequence“
multiple=“false“ optional=“false“>
</item>
```

With these two lines we give the following information to the **XCEL Processor**: There is an Item of type 'structuringItem'. 'StructuringItem' means that the Item does not model any semantic behaviour but just describes a well defined structure. Like every element in an XCEL Tree (except the Processings) the Item has an 'identifier' that can be used later to reference the Item. With the 'order' attribute we define in what order the child-elements of the Item must match the byte stream derived from the **inputfile**. In this case the Item prescribes that the containing children must appear in the sequence in which the elements are specified in the XCEL Tree definition. Since the

intended use of the Item is to describe the complete file we set the 'multiple' attribute to false. With the 'optional' attribute we express that the root Item must match to the input for being accepted as a proper description of our format. Note that the latter three attributes are set to their default values and are included here only to show explicitly the complete interpretation of the Item. Thus, the following description is equivalent to the one above:

```
<item xsi:type="structuringItem" identifier="ourFile">
</item>
```

In the next step we want to create a description for one single line of the file.

```
<item xsi:type="structuringItem" identifier="ourFile">
    <item xsi:type="structuringItem" identifier="oneLine"
        multiple="true" optional="false">
    </item>
</item>
```

Again we have prepared an Item to hold the description of one line. As we assume that our file contains at least one line (a file usually contains many lines) we set multiple to true and optional to false. The XCEL Processor will now try to apply this Item to the Inputfile as often as it matches. Because of the setting 'optional=false' the processor would abort with an error if the first line does not match to the description.
For the next step we have to formulate the concrete representation of one line.

```
<item xsi:type="structuringItem" identifier="ourFile">
      <item xsi:type="structuringItem" identifier="oneLine"
            multiple="true">
      <symbol identifier="oneCharacter"
            interpretation="ASCII" length="1"
            multiple="true">
            <nonValidValues>
                  <value>CR</value>
            </nonValidValues>
      </symbol>
      <symbol identifier="carriageReturn"
              interpretation="uint8" value="13"/>
      </item>
</item>
```

Now we have entered the next level of precision. Two Symbols were added to the line-Item. The purpose of the first Symbol is to consume all characters but not the carriage-return. The second Symbol is used to consume the carriage-return. With the newly introduced information any XCEL Processor should be able to parse through the complete file.

Let´s take a detailed look at the new elements. The 'oneCharacter'-Symbol defines an **interpretation**, which is ASCII in this case. In the XCEL we differentiate between encoding and interpretation. The interpretation of a certain element defines how the value of a processed value is matched to the lexical space. The encoding defines on what basis the addressing of a Symbol should be done. The default encoding behaviour is to interpret all length and start positions directly as byte offsets in the Inputfile. In this example this means, that the length is interpreted as exactly to be one byte long. If the encoding would be utf-8, a length of one would mean to read one character which must not necessarily correspond to one byte.
The Symbol is further described by the 'nonValidValues' tag. Within this tag we explicitly exclude the carriage-return from the set of matching characters. We can summarize the meaning of the statement with a sentence like this: “Read one byte from a raw byte stream as long as its ASCII-interpreted value is not carriage-return.”
The second symbol of the example is simply used to consume the carriage-return that was explicitly excluded by the symbol above.
Please note that if the first line was read, the setting 'multiple=“true“' of the 'oneLine'-Item forces the parser to apply the same procedure again as long as it matches.

Now we are able to read single lines but our goal was to extract surnames from the file. So the next step will be to differentiate between forenames and surnames in one line. This can be done by refining the definition of one line.

```
<item xsi:type="structuringItem" identifier="ourFile">
      <item xsi:type="structuringItem" identifier="oneLine"
            multiple="true">
            <symbol identifier="newCharacter"
                  interpretation="ASCII" length="1"
                     multiple="true">
                  <nonValidValues>
                        <value>SP</value>
                  </nonValidValues>
            </symbol>
            <symbol identifier="white space"
                  interpretation="uint8" value="32"/>
          <symbol identifier="oneCharacter"
                  interpretation="ASCII" length="1"
              multiple="true">
                  <nonValidValues>
                        <value>CR</value>
                  </nonValidValues>
            </symbol>
            <symbol identifier="carriageReturn"
                  interpretation="uint8" value="13"/>
      </item>
</item>
```

To distinguish between the forename and the surname we have added two new Symbols to our description. The first Symbol consumes all characters until a white space appears. The second Symbol consumes the white space. The rest of the description remains constant and is used to read the rest of the line.
With the 'oneCharacter'-Symbol we are now able to consume all surname characters. But for the extraction we have to add the additional information that these characters should be interpreted as being a surname.

```
<item xsi:type="structuringItem" identifier="ourFile">
      <item xsi:type="structuringItem" identifier="oneLine"
            multiple="true">
            <symbol identifier="newCharacter"
                    interpretation="ASCII" length="1"
                    multiple="true">
                  <nonValidValues>
                        <value>SP</value>
                  </nonValidValues>
            </symbol>
            <symbol identifier="white space"
                    interpretation="uint8" value="32"/>
            <item xsi:type="definitionItem"
                  identifier="asurname"
                  name="surname">
                  <symbol identifier="oneCharacter"
                          interpretation="ASCII"
                          length="1" multiple="true">
                        <nonValidValues>
                              <value>CR</value>
                        </nonValidValues>
                  </symbol>
            </item>
            <symbol identifier="carriageReturn"
                    interpretation="uint8" value="13"/>
      </item>
</item>
```

With the newly introduced Item we have added the information to the description that all parts of the line that are matching to the 'oneCharacter'-Symbol can be handled as one logical entity. Therefore the type of the surname Item is set to 'definitionItem'. The main difference between a 'structuringItem' and a 'definitionItem' is that the 'definitionItem' has a semantic meaning. This means that a matching 'definitionItem' is appended to the Result Tree while a 'structuringItem' is not. With introducing the 'definitionItem' the Result Tree does now contain an element we can refer to by its name. The value of this element in the **Result Tree** is defined by all values that have matched to the description within the 'definitionItem'.
With the description above we provide all information needed by the **XCEL Processor** to create a Result Tree with the intended information but we have not explicitly described what parts of the tree should be transferred into the (XCDL) output. This information must be provides in the part of the overall XCEL Schemas we call NamesLib [see Handling Names]. A NamesLib defines all names of properties that the parser should extract from a Result Tree. This means that the very last step would be to define a NamesLib that contains a property „surname“. After doing this, we can expect to get an XCDL output like this:

```
<?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" >
<property id="p1" source="raw" cat="descr" >
<name>surname</name>
            <valueSet id="i_i1_s1" >
                <labValue>
                    <val>Leibniz</val>
                    <type>string</type>
                </labValue>
                <dataRef ind="normAll" />
            </valueSet>
            <valueSet id="i_i1_s2" >
                <labValue>
                    <val>Baumgarten</val>
                    <type>string</type>
                </labValue>
                <dataRef ind="normAll" />
            </valueSet>
                  ...
        </property>
    </object>
</xcdl>
```

To get a detailed introduction how to build NamesLibs please refer to the „Extending the XCEL“ chapter. To see a full working implementation of this example refer to the „HowTo XCEL“ chapter.

1.3.1 Abbreviations, Definitions, Terms

1.3.2 Language specific terms

All terms which are used with a specific meaning when describing the XCEL are capitalized within this document

1.3.2.1 Content Model

In this document the term *Content Model* is used for the different ways the content of an Item can be structured in. This structure is specified in the Item's 'order' attribute. Valid values for it are 'all', 'sequence' and 'choice', each constituting a different Content Model.

1.3.2.2 Item

Items are language elements of the XCEL. Together with **Symbol**s and **Processing**s they establish the **XCEL Tree**. An Item is an entity that groups other entities in a structural, logical or semantic manner. An Item has always children; it is always non-terminal. The starting Item of any XCEL Document is called root Item.

1.3.2.3 Processing

Processings are language elements of the XCEL. Together with **Symbol**s and **Item**s they establish the **XCEL Tree**. A Processing encapsulates runtime dependencies that affect the XCEL Tree structure or the byte stream depending on the occurrence of single values.

1.3.2.4 Result Tree

The *Result Tree* stores matching **XCEL Element**s. While the **XCEL Tree** defines all potentially possible combinations of values that can occur within a file encoded according to a specific format, the Result Tree stores only the values which have actually been used in the encoding of one specific file. The Result Tree is the intermediate format between the parsing component and the output module of an **XCEL Processor**.

1.3.2.5 Symbol

Symbols are language elements of the XCEL. Together with **Item**s and **Processing**s they establish the **XCEL Tree**. A Symbol is a terminal entity that describes the position, length, encoding and semantics of a byte sequence.

1.3.2.6 XCEL Document

An *XCEL Document* is an XML file that is valid against the XCEL Schemas. It describes a particular file format. A 'PNG XCEL instance' is e.g. an XCEL document representing of the PNG file format. An XCEL Document consists of four parts, each of which contains one XCEL Tree [see The XCEL Main Structure]. An XCEL Document is effectively a program written in XCEL, which allows the extraction of characteristics from a specific file format.

1.3.2.7 XCEL Element

The term *XCEL Element* (short: 'element') is used when referring collectively to Items, Symbols and Processings.

1.3.2.8 XCEL Processor

An *XCEL Processor* is a software tool that is able to process an XCEL Document and apply it to an **inputfile** encoded according to the format described in that document. The XCEL Processor developed by Planets is called 'Extractor'.

1.3.2.9 XCEL Schemas

Entirety of the single XML schemas which together define the XML syntax of the XCEL. See XCEL Schema architecture.

1.3.2.10 XCEL Tree

The XCEL describes file structures as a tree. Each element [see **XCEL Element**] of the tree formulates facts about a file structure. An *XCEL Tree* is made up of **Item**s, **Symbol**s and **Processing**s. The root of an XCEL Tree must be an **Item**. Leaves of an XCEL Tree can either be **Symbol**s or **Processing**s. The XCEL Tree is the basic structure of an **XCEL Document**. For further readings see The XCEL Datamodel.

1.3.3 Common terms

1.3.3.1 digital object

The term *digital object* is used loosely within this document to discuss all kinds of items which reside on a computer and are understood as one logical entity. A digital object on a computer system can therefore be represented in many different ways: as a record, e.g., one file, two files, three files etc.
In most contexts of this document we need a more precise term: we speak of files, when we refer to digital objects which are encoded according to a specific format [see HowTo handle Multiple File Formats].

1.3.3.2 file format

A *file format* is a set of rules which formalizes the complete knowledge needed to process the information contained within a distinct and complete block of digital data, traditionally called a file. Modern information objects frequently divide the data they are made from between more than one file. In such cases one file format is understood to be describing the rules for all these blocks of digital data related to each other.

1.3.3.3 file

A *file*, or more precise a regular file, is the amount of data handled as a distinct set of bytes, referred to by a name and a set of characteristics administered by a specific part of the operating system, called file system. A file typically consists of two logically distinct blocks of data. The first data block is handled by the File System and is typically used to store administrative attributes like access permissions etc. The second block is a sequence of bytes that can be described by a certain **file format**. To describe the content of this second block is the task of the XCEL.

1.3.3.4 inputfile

The term *inputfile* is used in this documentation to refer to a file that is described by an **XCEL Document**.

1.3.3.5 file property / file characteristic

The XCEL supports the extraction of abstract *properties* of a class of files, encoded according to a specific file format. The value of a specific property extracted from a specific file is a *characteristic* of that file. It is not always clear in natural language, whether the plural “characteristics” refers to instances of the same property extracted a multitude of files or to one instance each of a multitude of properties extracted from one file. To make that difference absolutely clear in all instances, would have reduced the readability of this document considerably. We have, therefore, decided not to emphasize this difference, unless it is not apparent from the context.
In principle all **XCEL Element**s that have a 'name' attribute describe file characteristics. In the XCEL two types of elements support this attribute: all **Symbol**s and **Item**s using also the attribute type=”definitionItem”). To define that a certain XCEL Element constitutes a property, the name of the element *must* be defined in the NamesLib [see Handling Names], however.

1.3.3.6 lexical space / value space

The terms *lexical space* and *value space* are introduced to distinguish between the logical definition of a value (value space) and its physical representation within the XCEL language (lexical space). One of the objectives of the XCEL is to provide all necessary tools to map different lexical representations of a common logical value to a common lexical representation.

2. XCEL Elements

This section describes the tags which may occur within an XCEL specification. On their meanings and usage, see the examples in the various “HowTo” sections.

2.1 Items

2.1.1 Usage:

XCEL Items establish the inner elements of an **XCEL Tree**. Every Item must have at least one child of type Item, **Symbol** or **Processing**. Every XCEL Tree must begin with an Item as root element. Depending on their type, Items can be used for two things: On the one hand Items can be used to define the order of a sequence, on the other hand items can be used to define logical groups of other **XCEL Element**s to provide a unified access to complex content. The first kind of Items is of type 'structuringItem', the second one is of type 'definitionItem'.
With Items of type 'structuringItem' the language can define in which order the contained elements must appear to meet the requirements of the described file format. The usage of the 'order' attribute in a 'structuringItem' enables to define sequences with three different structures: A 'sequential' set is a sequence that prescribes that every single element in the sequence must match exactly in the order in which it occurs within the definition. To that effect the sequential definition of a set of elements A, B, C defines that the description of the elements must match in the order {A, B, C}. The attribute 'order' with value 'choice' applied to the same set defines that one of the elements A, B, C must match. That allows three possible sequences {A}, {B} and {C}. A general rule for choice definitions is that at least one element of the set must match. With the Content Model 'all' the language provides a third alternative to define sequences. The 'all' order applied to the set A,B,C defines six possible sequences {A,B,C}, {A,C,B}, {C,A,B}, {C,B,A}, {B,A,C} and {B,C,A}.
All sorts of sequences can be manipulated in detail with the attributes multiple and optional which can be applied to every XCEL Element.

2.1.2 Types:

2.1.2.1 structuringItem

A 'structuringItem' defines rules about the handling of a certain set of elements. In general a 'structuringItem' defines the order in which the child-elements must match to the byte stream. An Item of type 'structuringItem' is only used for parsing but never becomes a member of the Result Tree.

2.1.2.2 definitionItem

A 'definitionItem' is used to group elements to a logical unit. Usually 'definitionItems' have a name. For an example see [Introductory Example]. 'DefinitionItems' can exactly be used like 'structuringItems', the important difference to 'structuringItems' is that a matching 'definitionItem' will be added to the result Tree. A 'definitionItem' also provides an optional name attribute. If the name attribute is used, the complete sequence defined by the Item can be handled as one entity. Sometimes it is necessary to define that single format properties described by the XCEL are related to others but are still distinct characteristics. In such cases a 'definitionItem' without a name can be used to express that its child-elements form a group of related properties.

2.1.3 Child Elements:

Symbol, Processing
range , name

2.1.4 Explicit Attributes:

2.1.4.1 multiple

description
If multiple is set to true the element can appear multiple times. In general an element with 'multiple=”true”' will be applied as often as it matches the byte stream.
*Please note that bytes that are consumed once can not be shifted later to another element. E.g.: A regular expression like “.*t” cannot be processed within the XCEL. The dialect of regular expressions used within the XCEL requires the explicit exclusion of the trailing t like in “[^t]*t”.*
values
true , false
default value

false

2.1.4.2 optional

description
If 'optional' is set to 'true' the parent element can match even if the optional element did not; with other words, such an element may be missing. With optional elements one can formulate exceptions from the overall Content Model. If 'optional' is set to 'false' the intended behaviour depends on the Content Model.
values
true, false
default value
false

2.1.4.3 order

description
With the 'order' attribute an Item can define the Content Model for the sequence it is representing. Setting 'order' to 'sequence' means to apply the child-elements in the given order to the Inputfile. An order of 'choice' means that exactly one child-element must match to the current position in the bytestream. It is assumed that all elements are tested in the order of their appearance in the XCEL Tree. The 'all' Content Model is the most flexible model to use. It defines that all combinations of sequential concatenations of the child elements are possible. However like the two other Content Models, the all model is sensitive to the order in which the child elements are defined in the XCEL Tree. That means: the elements defined first are first tested against the Inputfile.
values
sequence, choice, all
default value
sequence

2.1.4.4 identifier

description
The XCEL currently requires all elements to define an identifier. An identifier must be unique in the whole **XCEL Document**. *For later versions it is planned that identifiers need only be applied to elements that are referenced elsewhere in the document*
values
All values that are allowed for the XML Schema ID data type.[5]
default value
-

2.1.4.5 externalSource

description
The 'externalSource' attribute can be used to switch to another file while processing the XCEL Tree.
values
An absolute local file path.
A file path relative to the originally processed file.
It is planed to generalize the path syntax to the URI format.
default value
-

2.1.4.6 internalSource

description
The 'internalSource' attribute can be used to describe the content of an element already read by a previous part of the XCEL Tree. This feature can be used to describe parts of the file with respect to different criterion. An example usage of the 'internalSource' feature is given in section HowTo Recursive format descriptions – Encoding Chains.
.
values
An value corresponding to the ID of an element already read.
default value
-

[5] David Peterson et al., XML Schema 1.1 Part 2: Datatypes, 2006. <http://www.w3.org/TR/xmlschema11-2/>

2.1.4.7 start

description
The 'start' attribute is used to predefine a start position for the element. The element will be applied to the defined position. Usually the definition of a start position is not necessary within the XCEL. In most cases start positions are calculated by the XCEL Processor or are set via XCEL Processings during runtime. Start positions can also be set in the range tag which is an optional tag that can appear within Items and Symbols.
values
All integer values between 0 and the length of the byte stream processed.
default value
-

2.1.4.8 length

description
The 'length' attribute is used to predefine the valid length of an XCEL Element. When using a length definition for Items the length value is used to restrict the application of its child-elements to a certain sub-sequence of the file. This can be useful in cases where a combination of Symbols is used to read a structure until a certain offset is reached.
values
Any natural number
default value
-

2.1.4.9 name

description
Items can define a name. In general the definition of names only makes sense for Items of type 'definitionItem'. However 'structuringItems' can define names as well. In the XCEL names are extremely important. In most cases names are directly related to properties. Two special names exist in the XCEL: The name 'normData' is used to refer to a “property” which describes the data contained in file as a characteristic of that file.[6] The name 'dummy' can be used for elements which are included to skip over certain parts of the file. Elements named 'normData' and 'dummy' are processed slightly different than others.
values
In principle all tokens are allowed. In practice it is a good idea to restrict the usage via an XML Schema to certain names [see XCEL Schema architecture].
default value
-

2.1.4.10 normDataRelation

description
The 'normDataRelation' attribute can be used to define that a certain property has a direct relation to the 'normData'. For example font styles have a direct relation to the 'normData' within text files.
values
true, false
default value
false

2.1.4.11 newObject

description
The 'newObject' attribute is used to define that an Item should be handled as a new object.
values
true, false
default value
false

2.1.4.12 print

description
The print attribute can be used to print the processing information about a single element during runtime.

[6] Please refert to Planets deliverable PC/2 D8: Sebastian Beyl, Volker Heydegger et. al., Final XCDL Specification, 2008.

values
true, false
default value
false

2.1.5 Implicit[7] Attributes:

2.1.5.1 interpretation

description
The 'interpretation' defines how values are mapped to the lexical space. For 'definitionItems' the interpretation is per definition the same as the interpretation of the first child element. Roughly speaking, interpretations describe the data type of an item.
values
Please refer to the schema definition.
default value
uint8

2.2 Symbols

2.2.1 Usage:

Symbols are atomic entities that have no logical children within the **XCEL Tree**. Therefore Symbols always are represented by leaves in the tree. In general, Symbols are used to define the encoding, interpretation and length of a certain byte sequence. Symbols can define that a certain amount of bytes should be consumed and interpreted in a specific way. Symbols can also formulate constraints for the values they can consume. The definition of constraints can be done in different ways. The values that a Symbol is allowed to consume can be defined explicitly by listing them in their interpreted form; Symbols can also define inclusion- or exclusion-lists as well as ranges within an interpreted lexical space; in the end Symbols can simply avoid to prescribe a value space. A Symbol matches up to a given position in the byte stream, if the consumed bytes are successfully converted into the lexical space the Symbol prescribes. With Symbols the XCEL provides a mechanism to read arbitrary data according to a specific interpretation, which describe valid characteristics of a file. The mechanism also allows to specify properties of a given format, which represent a characteristic of each file encoded according to that format (default characteristics, in other words). The XCEL Processor is responsible for finding all characteristics valid for a given file. The branches of an XCEL Tree that represent valid characteristics are appended to the **Result Tree**. If a Symbol does not match the byte stream when the XCEL Processor tries to apply it, the actions taken depend on the context (the parent Item and the qualifiers and quantifiers of the Symbol). With non-optional Symbols the formulation of conditions is possible, which refine the actions to be taken further. A non-optional Symbol that does not match at the current byte position usually indicates that its current XCEL branch description is not true for the given file, that the property expressed by the Symbol is not instantiated within that file, with other words.

2.2.2 Types

-

2.2.3 Child Elements:

range, value, validValues, nonValidValues, name

2.2.4 Explicit Attributes:

2.2.4.1 multiple

description
If multiple is set to true the element can appear multiple times. In general an element with 'multiple=”true”' will be applied as often as it matches the byte stream.
*Please note that bytes that are consumed once can not be shifted later to another element. E.g.: A regular expression like “.*t” cannot be processed within the XCEL. The dialect of regular*

[7] We call an attribute “implicit”, if it is not an attribute in the XML sense, but describes a quality of a tag, which is derived form XML attributes occurring within the child elements of the element in question.

*expressions used within the XCEL requires the explicit exclusion of the trailing t like in "[^t]*t".*

values
true, false
default value
false

2.2.4.2 optional

description
If 'optional' is set to 'true' the parent element can match even if the optional element did not; with other words, such an element may be missing. With optional elements one can formulate exceptions from the overall Content Model.
values
true, false
default value
false

2.2.4.3 identifier

description
The XCEL currently requires all elements to define an identifier. An identifier must be unique in the whole **XCEL Document**. For later versions it is planned that identifiers need only be applied to elements that are referenced elsewhere in the document.
values
All values that are allowed for the XML Schema ID data type.[8]
default value
-

2.2.4.4 start

description
The 'start' attribute is used to predefine a start position for the element. The element will be applied to the defined position. Usually the definition of a start position is not necessary within the XCEL. In most cases start positions are calculated by the XCEL Processor or are set via XCEL Processings during runtime. Start positions can also be set in the range tag which is an optional tag that can appear within Items and Symbols.
values
All integer values between 0 and the length of the byte stream processed.
default value
-

2.2.4.5 length

description
The 'length' attribute is used to predefine the valid length of an XCEL Element. The length of a Symbol defines how many tokens it is supposed to consume. In the normal case a token, most frequently a character, corresponds directly to a fixed number of bytes. However, the actual number of bytes consumed, depends on the combination of the encoding and the value of length.
values
Any natural number
default value
-

2.2.4.6 name

description
The 'name' attribute is an optional attribute that is used to specify that a Symbol has a certain meaning. Only elements with names can later be included in the output. If a symbol matches more than once the Result Tree will contain a list of Symbols all having the same name. If the entirety of all Symbols of the same name shall be available as one single element in the Result Tree, an enclosing Item of type 'definitionItem' can be used.
Two special names exist in the XCEL: The name 'normData' is used to refer to a "property" which describes the data contained in file as a characteristic of that file.[9] The name 'dummy' can be used

[8] David Peterson et al., XML Schema 1.1 Part 2: Datatypes, 2006. <http://www.w3.org/TR/xmlschema11-2/>
[9] Please refert to Planets deliverable PC/2 D8: Sebastian Beyl, Volker Heydegger et. al., Final XCDL Specification, 2008.

for elements which are included to skip over certain parts of the file. Elements named 'normData' and 'dummy' are processed slightly different than others.

values
In principle arbitrary tokens are allowed as names. For practical purposes one will almost always restrict the number of applicable ones via an XML Schema specifying the valid names [see XCEL Schema architecture].
default value
-

2.2.4.7 normDataRelation

description
The 'normDataRelation' attribute can be used to define that a certain property has a direct relation to the 'normData'. For example font styles have a direct relation to the 'normData' within text files.
values
true, false
default value
false

2.2.4.8 interpretation

description
The interpretation of a Symbol defines how the value of the Symbol should be interpreted. Roughly speaking, interpretations describe the data type of a symbol.
values
uint8, uint16, uint32, uint64, int8, int16, int32, int64, ASCII
default value
uint8

2.2.4.9 value

description
Symbols can be used to prescribe that a certain value has to be assumed, though not explicitly present in the file. This value can be defined using the value attribute. Actually the value attribute is just a short form to avoid the usage of the longer 'validValue' construction which takes place as a child-element within the XML syntax of the Symbol.
values
An arbitrary value
default value
-

2.2.4.10 encoding

description
With the encoding attribute the parser can be informed to interpret the definitions of a Symbol, e.g. length or value, not as multiples of bytes, but of tokens encoded according to some encoding scheme.
values
Currently only Utf-8 encodings are supported beside of byte wise encoding.
default value
byte

2.2.5 Implicit[10] Attributes:

2.2.5.1 filterChain

The 'filterChain' of a Symbol chains actions to manipulate the value the Symbol is representing. Filters can be added to a Symbol via the 'addFilter' method described in the Processings section.

[10] We call an attribute “implicit”, if it is not an attribute in the XML sense, but describes a quality of a tag, which is derived form XML attributes occurring within the child elements of the element in question.

2.3 Processings

With **Symbol**s and **Item**s the XCEL provides basic mechanisms to describe sequential structures which extract the XCEL Tree from a file and imply its conversion into a Result Tree. With *Processings* the XCEL introduces a simplified method model. The usage of Processings allows to manipulate the **XCEL Tree**, the **Result Tree** or the **XCEL Processor** at runtime. The manipulation of the XCEL Tree is necessary to allow the dynamic redefinition of lengths and start positions during runtime. The dynamic reconfiguration of the XCEL Processor can ease the writing of XCELs. For instance if an ASCII-based file format is described, the XCEL Processor can be configured to skip over meaningless parts (e.g.,white space) automatically. Processings share some similarities with methods in other programming languages. A processing method is directly related to a piece of code within the XCEL Processor and refers to it by the methods name. All methods return a boolean value which is true if the method was executed successfully or false if it was not.
A Processing element can have different types. A Processing of type 'pushXCEL' defines that the methods defined within the processing will manipulate an already existing **XCEL Element** elsewhere in the XCEL Tree. The type 'pullXCEL' is used to copy an XCEL Element from somewhere else in the tree to the position right behind the Processing. The 'pullXCEL' mechanism allows the reuse of existing XCEL Elements at different places. The 'configureParser' type is used to call methods that should manipulate the standard behaviour of the XCEL Processor.
A Processing can use an arbitrary set of methods. The absence of any method is only meaningful in the case of Processings of type 'pullXCEL'. In that case the referenced element will simply be copied to the location right behind the Processing. For all kinds of Processings the definition of multiple methods is allowed and can be useful.
Processings of both types 'pullXCEL' and 'pushXCEL' must refer to an XCEL Element that can then be manipulated with the related methods. Normally referencing is done by ID. However, the referencing by name is also supported. If the reference points to multiple elements the processing will be applied only to the *last* element with the ambiguous name or id that was read.
Processing methods usually require some parameters to be passed. The value of a parameter can be specified within in the XCEL document or can be a reference to the value of an element read earlier. In the latter case the XCEL Element must be referred by ID or by name. Again the value that will be used in the case of ambiguity is the value of the XCEL Element that was last read. It is also possible to subsequently refer to every element that is identified by an ID by using the 'listRef' attribute.

2.3.1 Types:

pullXCEL, pushXCEL, configureParser

2.3.2 Child Elements:

ProcessingMethod
param

2.3.3 setByteOrder

description
Allowed for Processings of type 'pushXCEL' or 'pullXCEL'. The 'byteOrder' defines how the XCEL Processor handles numerical values extended over more than one byte.
parameters
@1: bigEndian, littleEndian
return
true

2.3.4 setName

description
Allowed for Processings of type 'pushXCEL' or 'pullXCEL'. Allows to define a name for the referenced XCEL Element. Names that are defined in the NamesLib are handled as properties (implying a characteristic of the file currently processed).
parameters
@1: An arbitrary name.
return
true

2.3.5 setStartPosition

description
Allowed for Processings of type 'pushXCEL' or 'pullXCEL'. Defines the start position of an XCEL Element.
parameters
@1: A natural number. If the number is higher than the length of the currently processed byte stream, the element can never match.
return
true

2.3.6 setLength

description
Allowed for for Processings of type 'pushXCEL' or 'pullXCEL'. Defines the length of an XCEL Element. There are two kinds of 'setLength' methods, the first takes one parameter, the second takes two. The first 'setLength' method simply requires a file offset. The latter one calculates the length as a multiple of the length of a data type.
parameters
Variant 1:
@1: A valid file offset
variant 2:
@1: A number that defines how often the data type identified by the second parameter must fit into the elements range
@2: A data type.
return
true

2.3.7 setInterpretation

description
Allowed for Processings of type 'pushXCEL' or 'pullXCEL'. Sets the 'interpretation' of the referenced value.
parameters
@1: Any data type supported by the XCEL
return
true

2.3.8 setOptional

description
Allowed for Processings of type 'pushXCEL' or 'pullXCEL'. Sets the 'optional' attribute of the XCEL Element
parameters
@1: true , false
return
true

2.3.9 setMultiple

description
Allowed for Processings of type 'pushXCEL' or 'pullXCEL'. Sets the 'multiple' attribute of the XCEL Element.
parameters
@1: true, false
return
true

2.3.10 addFilter

description
Allowed for Processings of type 'pushXCEL' or 'pullXCEL'. Filters are natively defined procedures identified by a type and a name. The filter must be applied to the value of the XCEL Element before it returns its value. The application of the filter itself takes place when the XCEL Element is processed but is not part of the addFilter method.
parameters
@1: filtertype

@2: filtername
@3 – N: Filter specific parameters
return
true

2.3.11 isEqual

description
Returns whether two values are equal. One value is taken from the referenced XCEL Element.
parameters
@1: the value to compare with
return
true, false

2.3.12 setSymbolDoesNotMatchHandle

description
Allowed for processings of type 'configureParser'. The method allows the definition of an action that is executed whenever a Symbol does not match. The action is identified by passing its name as the first parameter of the method. The method is currently in use to skip over white space.
parameters
@1: skipwhitespace
return
true, false

2.3.13 setAddressingScheme

description
Allowed for Processings of type 'configureParser'. The method allows to change the addressing scheme used by the XCEL Extractor. Currently we support two schemes: byte and separator.
The schema is identified by its name passed as the first parameter. If separator is used, the method requires a second parameter which defines the byte values of the separator. The byte values must be separated by a single white space.
When the addressing scheme is changed the length definitions of XCEL Elements are interpreted as numbers of addresses.
parameters
@1: byte, separator
@2: the separator as series of byte values, separated by white space
return
true

2.3.14 goToLastAddress

description
In use for processings of type 'configureParser'. After executing this method the XCEL Processor should shift its reading position to the last address of the Inputfile. If the addressing scheme is set to 'byte-wise', the next byte to be processed is the last one of the byte stream currently processed. If the addressing scheme is set to 'separator' the byte immediately after the last occurrence of the separator is the next one to be processed.
parameters
No parameters
return
true

2.3.15 goToNextAddress

description
In use for processings of type 'configureParser'. The execution of this method can be used to step forward through the addresses of a file.
parameters
No parameters
return
true

2.3.16 goToPreviousAddress

description

In use for processings of type 'configureParser'. The execution of this method can be used to step backwards through the addresses of a file.
parameters
No parameters
return
true

2.3.17 getItemNumber

description
In use for type 'pushXCEL'. The method performs a query on the Result Tree. With its first parameter the ID of an Item which has been read before is passed. The second parameter asks for a child-element of the Item by its index. At the end the value of the identified child-element is set to the XCEL Element which is referenced by the Processing.
parameters
@1: The ID of an already read Item.
@2: An index to identify a child of the Item referenced by its ID.
return
true

2.3.18 setValue

description
In use for type 'pushXCEL'. Sets the value of a Symbol. This will normally be used to access Symbols with length zero, for which no value has been set so far, that is. 'setValue' can than be used to define the value of the Symbol.
parameters
@1: The value.
return

2.3.19 setIdentifier

description
In use for type 'pushXCEL' and 'pullXCEL'. Sets the identifier of the referenced element.
parameters
@1: an ID
return
true

2.3.20 setExternalSource

description
In use for 'pullXCEL' and 'pushXCEL'. Sets the 'externalSource' attribute of an element. An external source is currently defined to be a local file.
parameters
@1: A path to a local file.
return
true,false

2.3.21 setInternalSource

description
In use for 'pullXCEL' and 'pushXCEL'. Sets the 'internalSource' attribute of an element. An internal source is defined as the ID of an XCEL Element read before.
parameters
@1: An ID of an already read XCEL Element.
return
true

2.3.22 extract

description
In use for 'configureParser'. With the extract method a recursive call of the extractor can be performed. The characteristics extracted from the second file will be placed into a specified new file.
parameters
@1: Path to a local file that should be processed
@2: Path to a local XCEL that describes the file to be processed

@3: Path to the output file
@4: Name of the property used to connect the new generated object to the overall description
return
true, false

2.3.23 unzip

description
In use for 'configureParser'. The unzip method can only be used in the preprocessing section (<preProcessing>) of the XCEL. The method unzips a file and stores the result in a certain directory.
parameters
@1: Path to a directory used to store the unzipped data.
return
true, false

3. XCEL Schema architecture

3.1 Overview

As an XML-based language the XCEL makes use of an extensible XML Schema Architecture. Within the latter chapters we have introduced the basic principles of the XCEL. In addition this chapter deals mainly with the XML specific mechanisms that allow the easy creation of new **XCEL Document**s. The XCEL is an extensible language; this means that well defined interfaces and mechanisms for its extension must be provided by the language specification as well as by the **XCEL Processor**. The formal language definition of the XCEL is formulated as a set of XML Schemas. The XCEL Core Schemas define the basic structure of an XCEL Document and the values that are allowed at certain points of the XCEL. While this basic structure of the XCEL cannot be changed, all other parts can.
The most easily extensible part of the XCEL language specification is the part which defines the characteristics [see
file property / **file characteristic**] to be extracted and presented to the user. That schema module is called NamesLib. A NamesLib is an enumeration of documented names. Every XCEL Document is connected with a set of such enumerations. The names defined within the NamesLib can be used as names within the XCEL. When an XCEL Document is processed properly to its end, the XCEL Processor uses the names connected with that XCEL Document to query the **Result Tree**. All names defined in the NamesLib will then appear in the output. This extension does not require and modification of the XCEL processor.

A general extraction language must deal with many data types, as they are used to encode various domains of information, like images, texts or sounds. Conventional object oriented programming languages support mechanisms to define new data types in form of classes; XCEL hides that concept, by processing only 'native' data types. These 'native' data types can be described within the XCE language, however. This means however, that the XCEL Processor has to be modified to support the new data type.
As we have seen in the previous chapter, the XCEL does also provide a simple method-call via its Processing elements. Within the XCEL Schema architecture the extension of methods supported by the Processing element is also extensible. As in the case of data types the extension of methods implies that the XCEL Processor must also be extended with a corresponding code block that handles the newly introduced method.

The following sections will explain how the XML Schemas for the XCEL are designed with respect to the extensibility of data types, methods and names.

3.2 XML Schema Parts

The XML Schema definition of the XCEL consists of four core modules and one adapter schema. The XCELBasicStructure Schema defines the overall structure of the XCEL. The schema makes use of certain types to define what values are permitted to use within the individual extensible parts of the XCEL. Those types are implemented as forward references which means that the

XCELBasicStructure Schema is not a standalone valid schema definition but expects to be used meaningful by being included into another schema that implements the forward referenced types. Besides the XCELBasicStructure, the core module of the schema architecture does also provide a schema which holds valid names for methods that can be used within the Processing elements, a schema for the data types that can be used within the interpretation attribute of Symbols and a schema, that defines basic file properties that are not restricted to use for any specific file format domain. These schemas are called XCELBuildInMethods, XCELBasicDataTypesLib and XCLBasicNamesLib.
Please note that the latter one was named with a different prefix than the others. The reason for that is the character of this specific schema which acts as a connector to the XCDL. The schemas prefixed with XCEL provide the definition of the file characterisation extraction mechanism; the XCLBasicNamesLib provides the basic means to output the extracted characteristics in the XCDL.
Not strictly speaking part of the core module, the XCELBasicExtension schema combines the core schemas to a meaningful whole. The XCELBasicExtension schema provides a fully valid schema that implements the forward referenced types of the XCELBasicStructure and can be used as starting point for a new XCEL Document. It provides also a template for XCEL extensions. To extend the XCEL one can simply copy the XCELBasicExtension file and include one's own types, methods or names in the corresponding parts. The use of one schema that assembles the definitions from different schemas has also the advantage that an XCEL Document must only refer to one schema file which is responsible for the handling of namespaces and such things. The following figure shows the current status of our XCEL work. There exist elaborate XCEL Documents for four file formats of different complexity; each of these is connected to a specific schema extending the general vocabulary of the XCEL. Each of them, furthermore, is connected to an individual NamesLib for that individual file format even if there are overlaps between TIFF and PNG and between PDF and DOCX, as they could be described as two instances of image and text formats.

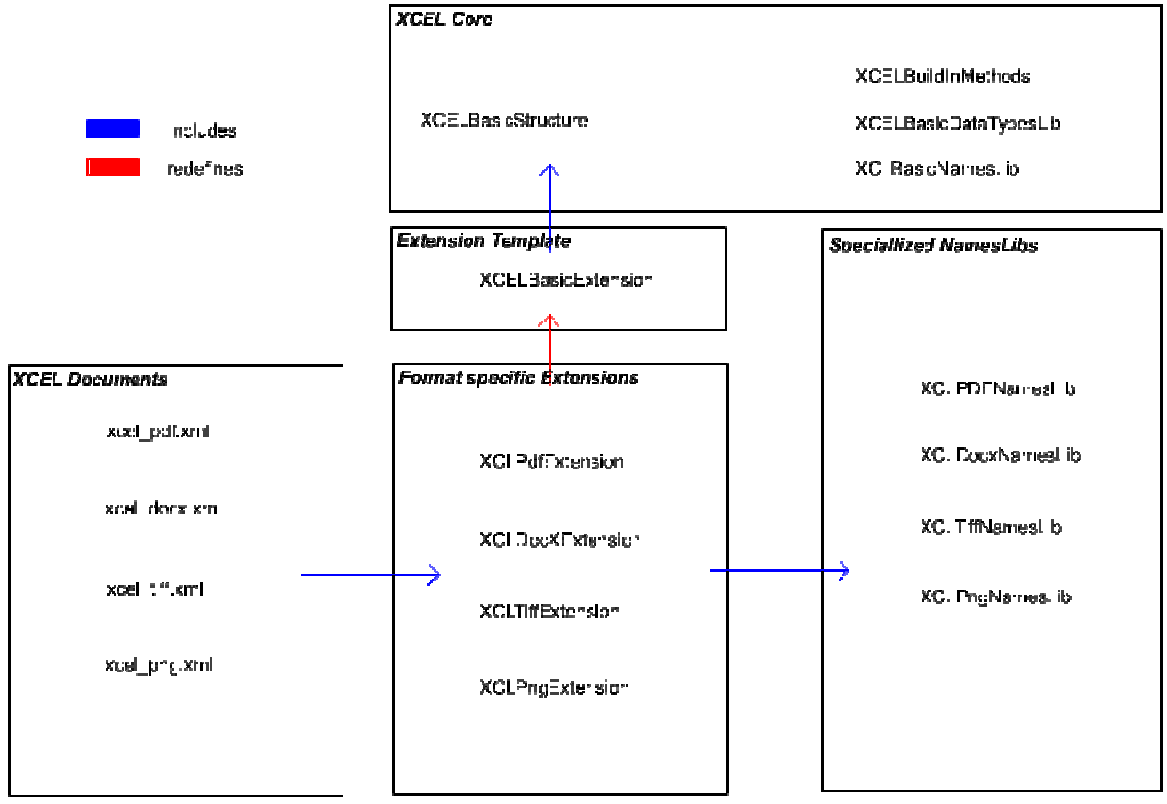

Figure 5: The XML Schema architecture of the XCEL

Please note that the currently existing extensions could be designed in a more hierarchical form. For example it would be possible to create one extension for text files and one extension for image files to store all common names. In any case we have decided to separate the extensions for the single file formats to avoid delays by depending on a generalized handling of the characteristics of file formats in general. Nevertheless this topic is a highly interesting field for further research and it will be covered within the Planets project by creating a general XCL Ontology that will provide a

more conceptual view on files than the current NamesLib does. A separate deliverable, representing the content of the current NamesLibs as an ontology has been submitted as a starting point for that further work. Please read the following chapter (see Extending the XCEL) for a description about the current status of name handling; name handling is also the subject of the next section.

3.3 Handling Names

This section links the development of the two connected XCL languages. While the XCEL describes how various file characteristics can be extracted from files of a specific format, the XCDL provides an abstract structure to store those characteristics in a normalised form and make them accessible under unified names. This is important in case one wants to create a new **XCEL Document** for e.g., an image format. ´Doing so, we have to decide how to name the single characteristics which are expressed in a specific way within the humanly readable documentation of the file format which we, however, want to extract in such a way, that they can be compared to characteristics which are logically the same, but potentially named differently within another file format. Because there are already some image formats described, it would be a good idea to stay compatible to the existing naming conventions. The problem one must face is the translation of the different vocabularies used by the maintainers of file formats into one common language. Here we can distinguish between different kinds of problems.
In the simplest case two file formats use just different terms for the same thing, e.g. TIFF uses 'imageLength' where PNG uses 'imageWidth' (the fact that TIFF stores the length usually as 16 Bit unsigned integers, while PNG uses 32 Bit unsigned integers is negligible once the values are interpreted).
Another sort of problem occurs if two file formats are using the same term for characteristics which are stored completely differently. A 'colour table', which is indeed always functionally a 'colour table' can be stored as an array of rgb-triplets or as three separate arrays, one for each RGB colour band, while some formats may also provide for lookup tables allowing more than three colour bands, e.g. an alpha channel, and so on. In such cases the terminology provided by the NamesLibs must at least be aware of the existence of such differences; for all practical purposes, it should define a standard representation of the data describing such a property, so the different representations in the individual file formats can be converted into that standard representation during the extraction of the characteristic.
The third, most severe, class of problems is introduced when a format defines workarounds to handle features that were not taken into account when the format was originally designed: so can, e.g.,the handling of alpha channels in layout based formats be imitated by mixing the colours of overlapping graphical elements in a well defined manner without introducing an explicit alpha channel into the colour space of the single elements.
As mentioned before, this specification, based upon the current NamesLibs cannot provide a general solution for these problems. But for further development a generalisation of the current NamesLib mechanism into an ontology-based mechanism, relating the way in which individual file formats store their properties to a generalized understanding of these properties within their respective domain, is indispensable.

4. Extending the XCEL

The currently defined XCEL can be extended by

- Creating a new XCEL Document
- Creating a new XCEL NamesLib
- Extending the XCEL Basic Structure
- Writing plugins for the XCEL Processor

4.1 Creating a new XCEL Document

An **XCEL Document** is a description of a file. Usually a file has a certain type, e.g. text, image, audio etc. One assumption of the XCEL is that objects of the same type are using the same concepts. For that the XCEL uses a set of controlled and documented terms to give the same things the same name. These terms are defined in XML Schemas we call NamesLib. To support

the creation of an XCEL instance we recommend to restrict the **XCEL Element**s defined in the XCELBasicStructure.xsd to contain only names that are described in the NamesLib of their format. This can be done by defining and extending the basic types in a file with the naming convention XCEL[*format*]Extension.xsd. We recommend to create new XCEL instances by associating them with the format specific XML Schemas. Thus your XML validator should be able to check the structure AND the used terms.
To describe the PNG format in XCEL you should start with:

```
<XCELDocument
xmlns="http://www.planets- project.eu/xcl/schemas/xcelstructure"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="http://www.planets-project.eu/xcl/schemas/xcl
      schemas/XCELPngExtension.xsd">
```

4.2 Creating a new XCEL NamesLib

A NamesLib is basically a list of terms that represents concepts that are used by a certain file format.

4.2.1 Creating a NamesLib by Hand

The first you should do when creating a NamesLib is to define the namespace and to include the BasicNamesLib by writing

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
       xmlns="http://www.planets-project.eu/xcl/schemas/xcl"
       xmlns:tns="http://www.planets-project.eu/xcl/schemas/xcl"
       targetNamespace="http://www.planets-project.eu/xcl/schemas/xcl"
       elementFormDefault="qualified">
<xs:include schemaLocation="XCLBasicNamesLib.xsd"/>
```

The next is to introduce an enumeration for the terms you want to use. We prefer to give our terms the basic type xs:string.

```
<xs:simpleType name="xclImageBasicNames">
<xs:restriction base="xs:string">
      <xs:enumeration value="1931IEC_ChromaticityRedX">
            <xs:annotation>
                  <xs:documentation>
                        value x of pair xy specifying red
                        colour according to 1031 IEC
                  </xs:documentation>
            </xs:annotation>
      </xs:enumeration>
      <xs:enumeration value="1931IEC_ChromaticityRedY">
            <xs:annotation>
                  <xs:documentation>
                        value y of pair xy specifying red
                        colour according to 1031 IEC
                  </xs:documentation>
            </xs:annotation>
      </xs:enumeration>
      ...
</xs:simpleType>
```

Example 6.3. (XCLImageNameslib.xsd) - Creating a NamesLib: Defining a new type

To support a more dynamic behaviour of your validator/editor you can group your terms in different schema types. But we recommend to provide one union that contains all types.

```
<!--********** union type for all image specific xcl names **********-->
<xs:simpleType name="xclImageNamesDefinitions">
```

```
      <xs:union memberTypes="nm:xclImageBasicNames
            nm:xclImageExtendedNames nm:xclImageValueLabels
            nm:xclBasicNameDefinitions"/>
</xs:simpleType>
```

4.2.2 Creating NamesLibs with Database support

During our work we have observed that the manual creation of the NamesLibs made it hard to keep track of the introduced terms. Therefore we have decided to administer all terms in one database which then can be used to generate NamesLibs from.

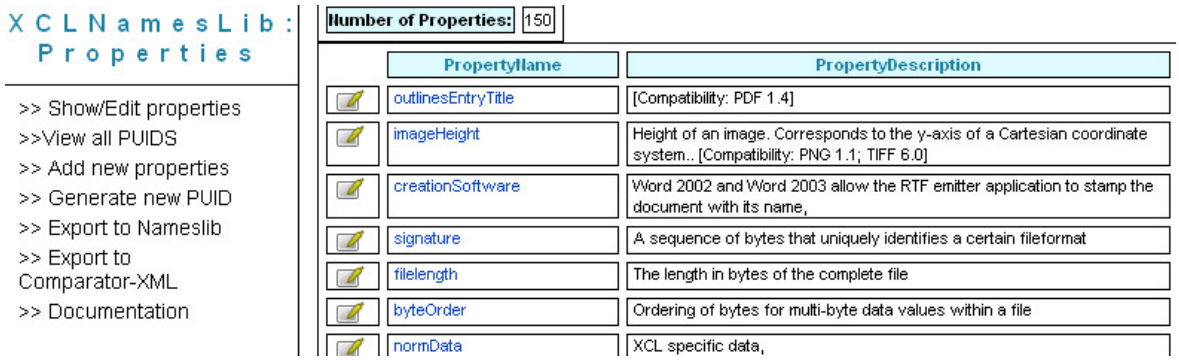

Figure 6: Interface for the NamesLib DB

When creating a new XCEL Document the first step after reading the format specification is to add the properties one want extract from the format into the database. During this process it is a good idea to check whether similar properties already exist in the list. When this is finished, a list of properties can be assembled and stored in association with a PUID.[11] Stored lists can then be exported as a NamesLib. The following sections describe how to make use of the NamesLib.

4.3 Extending the XCEL BasicStructure

To make use of a new NamesLib one has to extend the definitions of at least two types. A sample file for creating an extension is provided with the XCELBasicExtension.xsd. The new extension should follow the name convention XCL[format]Extension.xsd (where [format] refers to a specific file format: png, wav, etc). The next step is to define the namespace, to include your NamesLib and to include the XCELBasicStructure.xsd by writing:

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
          xmlns="http://www.planets-project.eu/xcl/schemas/xcl"
          xmlns:tns="http://www.planets-project.eu/xcl/schemas/xcl"
          targetNamespace="http://www.planets-project.eu/xcl/schemas/xcl"
          elementFormDefault="qualified">
        <xs:include schemaLocation="XCELBasicStructure.xsd"/>
        <xs:include schemaLocation="XCELImageNamesLib.xsd"/>
```

Now you can define that the terms from your NamesLib can be used as names and labels by writing:

```
<xs:simpleType name="extendedNameDefinitions">
        <xs:union memberTypes="
                  tns:xclBasicNameDefinitions
                  tns:xclImageNamesDefinitions"/>
</xs:simpleType>
```

[11] Pronom Unique Identifiers <http://www.nationalarchives.gov.uk/aboutapps/pronom/puid.htm>

This allows you to use terms from the XCLBasicNamesLib and the format specific terms from your NamesLib (here: tns:xclImageNamesDefinitions). You can also define what values should be used in your XML instance document by modifying the extendedValueDefinitions.

```
<xs:simpleType name="extendedValueDefinitions">
                <xs:union memberTypes=" xs:string xs:int"/>
</xs:simpleType>
```

Adding new datatypes or new methods to the XCEL can be done by defining the types Example (XCELImageExtension.xsd)

```
<xs:simpleType name="interpretationType">
      <xs:annotation>
            <xs:documentation>
                  The interpretation Type defines which
                  datatypes are available for symbols and
                  properties.
            </xs:documentation>
      </xs:annotation>
      <xs:union memberTypes="tns:xclDataTypeDefinition"/>
</xs:simpleType>
<xs:simpleType name="methodType">
      <xs:annotation>
            <xs:documentation>
                  The Method Type is used to define the known methods
                  that can be used in the processing element. Methods are
                  used for manipulating the XCEL Tree or the binary Data,
                  that is described by the XCEL. A Method must have a
                  corresponding Plugin in the XCEL Processor.
            </xs:documentation>
      </xs:annotation>
      <xs:union memberTypes="tns:basicMethodType"/>
</xs:simpleType>
```

New data types should be defined in a file with the name convention XCL[format]DatatypesLib.xsd. New methods should be defined in a file with the name convention XCEL[format]BuildInMethods.xsd.

5. XCEL Processor Rules

5.1 Introduction

In this clause we give some requirements and recommendations for **XCEL Processor** behaviour. The first subsections describe how a single value is handled on its path from the **inputfile** to the outputfile. The second subsection is a guideline for the handling of the single **XCEL Element**s in its processing context.

5.2 General Processing Chain

The general processing chain that is introduced in this section describes the travel of a single value from its beginning in the Inputfile to its end in the outputfile. The sections headlined with Querying and Writing assume that the output is written in an XCDL-like format.

5.2.1 Encoding

The first necessity when processing a digital file is to decode the bit stream. Usually this means to summarise bit sequences to meaningful parts and make them addressable as single entities. The most often used encoding is the byte-wise encoding which summarizes eight bits to one logical

unit, the byte. Other higher level forms of encodings are text encodings like cp1252 or utf-8. Those encodings are typically constructed on top of the byte level. The standard encoding of the XCEL is the byte-wise encoding. A typical characteristic of encodings is that encodings can be chained together, e.g. an utf-8 decoding works on a byte-wise interpreted file. Within the XCEL an attribute *encoding* is provided to inform the XCEL Processor that a certain encoding was used to store the information. By adding the information about the encoding the XCEL programmer tells the XCEL Processor that he wants address entities of the encoding e.g., if the encoding is utf-8 the reading of a Symbol with length *one* means to read one utf-8 character which not necessarily is the same as reading one byte.

5.2.2 Adressing

When a file or a part of a file is successfully decoded, the XCEL Processor must be able to address certain entities of the decoded stream. In any way the addressing depends on the underlying decoding.

5.2.3 Interpretation

The interpretation takes place when a set of entities is identified by encoding and addressing. An interpretation defines how the entities that are addressed should be interpreted. In principle the values an interpretation can have are very similar to native data types of a programming language. To give an example: if a Symbol of length *two* within a byte-wise encoded Item is interpreted as a uint16, that means that the XCEL Processor must handle the two values as one unsigned integer according to the current byte-order. An alternative processing could be introduced by setting the encoding to uint16 and define a length of one.
We can modify our example in the way that the Item defines an utf-8 encoding. The interpretation as uint16 would mean that we interpret two characters as one number with special characteristics (natural number, with 0 < number < 65536).

5.2.4 Filtering

In theory it is possible to chain more than two encodings or interpretations together. The 'addFilter' method used within Processings is one possible point to define encoding chains. Currently the so called 'filterchain' is used to perform filter actions like deflate or interlace on a certain value. In any case the defined filter must be applied to the value before it can enter the next step of processing.

5.2.5 Matching

If a value from the Inputfile has passed the chain of interpretation successfully, it must be comparable to a predefined value. That means that all values must be mapped to a lexical space that is representable in XML. An interpreted value matches to the predefined value of a Symbol if it is mapped to the same lexical representation as provided by the Symbol definition. Symbols can define a mapping between a value and a standard name. This allows to define a format specific value to be interpreted as a standardized term.

5.2.6 Querying

All matching elements of the XCEL will be stored in one data structure called the Result Tree. The tree does not necessarily just contain file-characteristics, it can also contain a lot of format-internal elements. With the NamesLib a user can define what values within the tree should be interpreted as file properties. For that the XCEL Processor must query the Result Tree with the property names defined in the NamesLib. The outputfile will only contain matching values those names were mentioned in the NamesLib.

5.2.7 Writing

If the XCEL Processor has decided which values to present in the outputfile there are still a few questions to answer. What type does the value have? Are their relations to other values? Is the value related to the normData?
The type of a value depends on the last applied interpretation within the encoding chain e.g., if an ASCII encoded value was interpreted as an integer the type of the value is integer; - if a byte-wise encoded value was interpreted as an ASCII character, the type is ASCII.

Some file characteristics are in any case related to other characteristics. To define such relations, the XCEL provides the construction of nameless definitionItems as well as the addRelation method. Both tools can connect single values. The output module of an XCEL Processor must take relations into account.
Values can be connected to other values but values can also be connected to certain parts of the normData. If a value was marked as a normData-related value, the XCEL Processor must analyse to what parts of the normData it belongs. This can usually be done by analysing the position of the value within the Result Tree.

5.3 XCEL States

The state of an XCEL Processor is defined by the current element E, its context C and the underlying byte sequence BS. State {E,C,BS} An XCEL element has

1. a *type*
   enumeration: {ITEM_TYPE, SYMBOL_TYPE, PROPERTY_TYPE, PROCESSING_TYPE}
2. a *quantifier*
   bool:isMultiple
3. it *can be optional*
   bool:isOptional
4. it *can be valid* (match to the binary stream)
   bool:isValid

The context of an element is given by its parents (Items). They can define

1. a *range* (byte positions in the binary stream)
   unsigned int:start (default 0)
   unsigned int:end (default the filelength)
2. a *Content Model*
   enumeration: {all,choice,sequence}

5.4 General Rules

A rule describes how an XCELProcessor should behave if a state {E,C,BS} is given.

5.4.1 For contexts with Content Model all

E = an XCEL Element
C = an Item with the attribute order=”all”
BS = somewhere in a file

- Any sequence constructable with the child-elements is valid.
- The XCEL Processor stays in the context as long as at least one element is available in the context and at least one element matches the next characters.
- The processor tries to apply the defined elements in its given order. If the processor has applied the last child-element, it begins with the first available element
- A child-element with 'isMultiple=false' can occur only one time in the sequence and will be removed from the context if it has matched.
- A child-element with 'isMultiple=true' can occur multiple times in the sequence. If the element matches the processor must check if it matches again to the next position. The element will never be removed from the context.
- A non-optional element must occur in at least one iteration. An iteration starts at the first element that was defined in the XCEL Document and ends at the last element. During an Iteration elements can be removed from the context.
- The processor escapes from the context if no element matches.
- The processing fails if one or more of its non-optional child-elements haven't matched.
- Elements with the 'name=”dummy”' cannot be applied more than one time in a single iteration. If an element with 'isMultiple=true' and the name 'dummy' appears the element will remain in the context but the XCEL Processor should not try to apply the element again to the byte stream until not all other available elements were tested.

5.4.2 For contexts with Content Model 'sequence'

E = an XCEL Element
C = an Item with the attribute order=”sequence”
BS = somewhere in a file

- The child-elements must match in the given order as defined by the XCEL Document
- If an XCEL element with quantifier 'isMultiple=true' occurs the processor has to check if its repetition matches again to the byte stream. The element is tested as long against the byte stream as it matches. The element can only repeat as often as the range prescribes.
- An element without quantifier 'isMultiple' can only match one time in the given context.
- A non-optional element must occur at the given position. Otherwise the context does not match.
- The processor escapes the context if all elements are successfully processed or if a non optional element does not match.
- The processing fails if either one of its child-elements doesn't match at its specified position.

5.4.3 For contexts with Content Model 'choice'

E = an XCEL Element
C = an Item with the attribute order=”choice”
BS = somewhere in a file

- One child-element must match
- If an XCEL element with quantifier 'isMultiple' occurs the processor has to check if its repetition matches to the byte stream again. The Element can only be repeated as often as the range prescribes it.
- The optional attribute is ignored.
- The processor escapes the context as soon as one element - and possibly its repetitions – matches.
- The processing fails if no child-elements matches the byte stream.

6. HowTo XCEL

6.1 HowTo XCEL PNG – defining a list of unordered items

The first format we have described in XCEL was PNG[12]. PNG is an image file format for presenting images over the web. The advantage of this intended usage is that therefore the PNG structure can be read and also described sequentially. The diagram below in Figure 7 provides an overview about the PNG structure. PNG files are organized in chunks where every chunk has in principle the same structure. PNG defines only a few rules about the order of chunks[13]. For example the header chunk must be stored at the beginning of the file, and the END-Chunk must be stored at the very end of the file. Between this two chunks all other chunks can occur. For parsing the file we can ignore the order of the chunks in between the header and the end because we don´t want to validate the file, but extracting properties from it.

[12] An extensive example of the XCEL application on PNGs is shown at <http://planetarium.hki.uni-koeln.de/XCL>
[13] W3C PNG Specification. <http://www.w3.org/TR/PNG/#5ChunkOrdering>

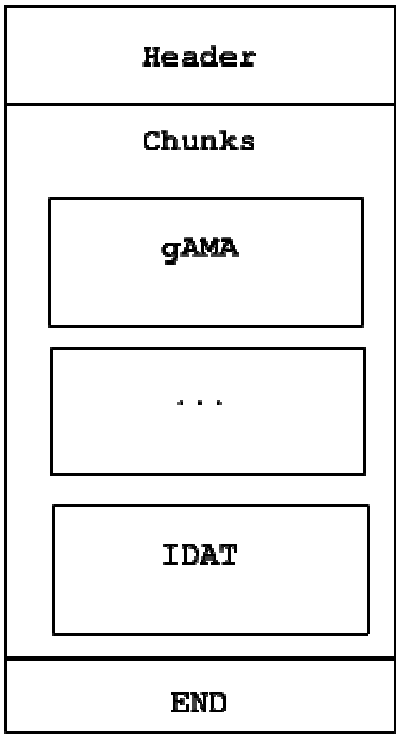

Figure 7: PNG file structure

This example should show how to define the sequence of chunks in the middle of a PNG file. At this point we can only adumbrate the XCEL for the whole PNG file. For the complete solution please refer to the XCL website[14].

```
<item identifier=”png” xsi:type=”structuringItem” order=”all”>
      <item identifier=”pngHeader”
            xsi:type=”structuringItem” >
      </item>
      <item identifier=”pngIDAT” xsi:type=”structuringItem”
            multiple=”true”>
      </item>
      <item identifier=”dummyItem”
            xsi:type=”structuringItem” multiple=”true”
            name=”dummy”>
      </item>
</item>
```

What the example shows is the definition of the overall chunk structure of a PNG file. With the order=”all” in the first Item we express that the contained elements can occur in any order. The **XCEL Processor** will stay as long in the context of the Item as one of its children matches the next position in the byte stream. The first child that is defined within the Item is again an Item introduced to hold the description of the PNG header. Note that the 'pngHeader'-Item needs to occur exactly one time because we have neither set 'multiple=”true”' nor 'optional=”false”'. If the 'pngHeader'-Item has been matched one time it will be removed from the context of the *png Item*. The next Item was introduced to provide the definition of an IDAT chunk. The IDAT chunk can occur multiple times in one PNG therefore the attribute 'multiple=”true”' was defined. That means the 'pngIDAT'-element stays in the context of the 'png'-Item even if it has been matched. The last Item is introduced to skip over chunks that are not supported by the description. The name 'dummy' indicates this special meaning and slightly changes the behaviour of the processor for this case. While usually the 'multiple=”true”' is interpreted as: “use this as long as it matches”, in the case of a dummy Item the meaning is: “use this but if it matches test all other alternatives before you use the

[14] <http://planetarium.hki.uni-koeln.de/XCL>

element again". This behaviour allows to define skipping elements that can occur multiple times without skipping the rest of the file once they have been encountered.
The example above does only provide the skeleton of a PNG XCEL. In the next example we want to show how the 'pngIDAT'-item can be represented. But first we should explain how a PNG chunk is defined in general.

Figure 8: PNG chunk

Figure 8 above shows the general structure of a PNG chunk. The meaning of the single fields is described shortly as:
Length: a four byte unsigned integer representing the length of the Data field.
Type: a four byte long identifier used to define what the Data field is about.
Data: a Data field of arbitrary length
CRC: a four byte long calculated checksum of the three other fields

The XCEL description for the chunk structure is given in the next example:

```
<item identifier="pngIDAT" xsi:type="structuringItem"
      multiple="true">
      <symbol identifier="chunkDataLength"
              interpretation="uint32" length="4"/>
      <symbol identifier="pngIDATIdentifier"
              interpretation="ASCII"
              optional="false" value="IDAT"/>
      <processing type="pushXCEL" xcelRef="normDataSymbol">
            <processingMethod name="setLength">
                  <param valueRef="chunkDataLength"/>
            </processingMethod>
      </processing>
      <symbol identifier="normDataSymbol"
            interpretation="uint8" name="normData"/>
      <symbol identifier="crc" length="4"/>
</item>
```

The example above shows the definition of the IDAT chunk. The first symbol reads four bytes and interprets them as one 32bit unsigned integer. According to the PNG specification the resulting value must to be interpreted as the length of the chunk data field. The second symbol defines the chunk type. All chunks are identified by their chunk type. The XCEL Processor uses the information of a predefined value to identify the proper branch within the 'png'-Item. The third element is a processing element used to set the length of the chunk data field according to the length field. The processing is of type 'pushXCEL' which means that it manipulates an **XCEL Element** that already exists in the **XCEL Tree**. The attribute 'xcelRef' refers by ID to an XCEL Element to manipulate it. In this case the referenced element is placed directly behind the 'processing' element. The processing calls a predefined method by its name, 'setLength'. The concrete length is committed by a parameter that refers in its 'valueRef' attribute to the Symbol that is supposed to contain the proper length. After applying the processing, the XCEL Processor enters now the next Symbol and set its value with respect to the calculated length. The last symbol simply reads the last four bytes of the chunk. The 'pngIDAT'-Item provides a general description of a png chunk and with one small

change it can be used for the 'dummyItem' in the latter example. In the case of the 'pngIDAT'-Item the 'normData'-Symbol has the reserved name 'normData' which defines that the concatenated content of all symbols with this name represents the basic data contained within the file. To define the 'dummy', symbol one must merely remove the name attribute from the symbol definition. For further reading about the handling of 'dummy' and 'normData' see the XCEL Processor Rules section.

6.2 HowTo write Sequential Description

Sequential descriptions of a file structure are the most common application of XCEL and for a large number of file formats they are also sufficient to extract the characteristics of files encoded in that format. The following example shows how to describe a simple date format.

Format:

```
04.04.2008
```

Description:

```
<item identifier="date" xsi:type="structuringItem" order="sequence">
      <symbol interpretation="ASCII" length="2" name="day" />
      <symbol interpretation="ASCII" value="."/>
      <symbol interpretation="ASCII" length="2" name="month" />
      <symbol interpretation="ASCII" value="."/>
      <symbol interpretation="ASCII" length="4" name="year" />
</item>
```

6.3 HowTo express Conditions

In procedural languages conditions usually take the form of if-else constructs. In the XCEL conditions must be modelled as branches of the **XCEL Tree**. The decision what branch is used is made by the **XCEL Processor** which therefore must analyse the current values of the branches. The next example should demonstrate this XCEL feature. The following format shows an ASCII string where different parts have different meanings. The parts marked in red indicate that a number of a certain length follows. The string '4Digits' is always followed by a four digit number while the string '2Digits' is always followed by a two digit number.

Format:

```
4Digits12342Digits562Digits784Digits9123
```

Description:

```
<item identifier="date" xsi:type="structuringItem" order="choice"
multiple="true">
      <item identifier="4Digits" xsi:type="structuringItem" >
            <symbol interpretation="ASCII" value="4Digits"/>
            <symbol interpretation="ASCII" length="4"/>
      </item>
      <item identifier="2Digits" xsi:type="structuringItem" >
            <symbol interpretation="ASCII" value="2Digits"/>
            <symbol interpretation="ASCII" length="2"/>
      </item>
</item>
```

The construction can be explained as follows. We start with an **Item** of order choice which is allowed to occur multiple times. The XCEL Processor must now decide which element from within the choice-Item holds the proper description for the string at the current parsing position. The first **Symbol** in the first Item requires that an ASCII-interpreted byte sequence must match to the value 4Digits. If it does the next Symbol consumes four ASCII values. If it does not match the XCEL Processor will switch to the next element within the choice context. By setting the multiple attribute of the choice-Item the selection process will be repeated until the end of file is reached.
As in other languages their exist many ways to express simple formats like the one given above. The following description shows an alternative XCEL for the given format.

Description:

```
<item identifier=”date” xsi:type=”structuringItem” order=”all”>
      <item identifier=”4Digits” xsi:type=”structuringItem”
            multiple=”true”>
            <symbol interpretation=”ASCII” value=”4Digits”/>
            <symbol interpretation=”ASCII” length=”4”/>
      </item>
      <item identifier=”2Digits” xsi:type=”structuringItem”
            multiple=”true”>
            <symbol interpretation=”ASCII” value=”2Digits”/>
            <symbol interpretation=”ASCII” length=”2”/>
      </item>
</item>
```

In this case the description starts with an Item of Content Model all. The 'order=”all”' attribute defines that all descending elements can occur at any position of the format. Both Items within the 'all-Item' are set to be allowed to appear multiple times. Otherwise an item that matches one time would be removed from the context.
Their is one last possibility to model the format. The next example shows how to describe the format with an sequence Item.

Description:

```
<item identifier=”date” xsi:type=”structuringItem” order=”sequence”
      multiple=”true”>
      <item identifier=”4Digits” xsi:type=”structuringItem”
            multiple=”true” optional=”true”>
            <symbol interpretation=”ASCII” value=”4Digits”/>
            <symbol interpretation=”ASCII” length=”4”/>
      </item>
      <item identifier=”2Digits” xsi:type=”structuringItem”
            multiple=”true” optional=”true”>
            <symbol interpretation=”ASCII” value=”2Digits”/>
            <symbol interpretation=”ASCII” length=”2”/>
      </item>
</item>
```

The description defines a fixed sequence but both elements of the sequence are optional. The sequence itself can occur multiple times and matches as long as one child of the sequence matches (general rule).

6.4 HowTo model Value Constraints

The definition of value constraints is a very important feature of an Extraction Language. The XCEL provides many methods to formulate such constraints. There are two different constructs that can be used to define a value space. On the one hand one can define explicitly all values a certain **Symbol** can take, on the other hand the definition of all values a Symbol can *not* take is possible too. For string processing regular expressions are the most powerful way to describe a certain format.
The example used here is about a format that provides a comma separated list of numbers.

Format:

```
      12345,122345,13,14567,3456
```

Description:

```
<item identifier=”date” xsi:type=”structuringItem” order=”choice”
      multiple=”true”>
      <item identifier=”4Digits” xsi:type=”defintionItem” name=”aNumber”>
            <symbol identifier=”aDigit” interpretation=”ASCII” length=”1”
                  multiple=”true”>
                  <nonValidValues>
```

```
                        <value>,<value>
                  </nonValidValues>
            </symbol>
      </item>
      <symbol identifier=”aComma” interpretation=”ASCII” value=”,”/>
</item>
```

To understand the main structure please refer to the “HowTo express conditions” section. In this section we want to focus on the aDigit-Symbol. The Symbol is used to read all digits of one number. The Symbol accepts all values but not a comma. While this definition is clearly a simplification of the “format specification” it is fully adequate to parse the format.
A more precise description is given in the next example.

Description:

```
<item identifier=”date” xsi:type=”structuringItem” order=”choice”
      multiple=”true”>
<item identifier=”4Digits” xsi:type=”defintionItem”  name=”aNumber”>
            <symbol identifier=”aDigit” interpretation=”ASCII”
                  length=”1” multiple=”true”>
                  <validValues>
                        <range>
                              <start>0</start>
                              <end>9</start>
                        </range>
                  </validValues>
            </symbol>
      </item>
      <symbol identifier=”aComma” interpretation=”ASCII” value=”,”/>
</item>
```

The example above shows an explicit definition of the Symbols value space. Within the 'validValues' tag a range of values is given. In this case the values 0 and 9 describe a well defined set of ASCII characters (the decimal digits) which are between the two values specified. An equivalent description can be given by adding the single values to the value space:

```
<validValues>
      <value>0<value>
      <value>1<value>
      <value>2<value>
      <value>3<value>
      <value>4<value>
      <value>5<value>
      <value>6<value>
      <value>7<value>
      <value>8<value>
      <value>9<value>
</validValues>
```

In the case of text based formats values can also be described with a regular expression like shown in the example below.

Description:

```
<item identifier=”date” xsi:type=”structuringItem” order=”choice”
      multiple=”true”>
      <item identifier=”4Digits” xsi:type=”defintionItem”
            name=”aNumber”>
            <symbol interpretation=”ASCII” length=”1”
                  matchingBehaviour=”regexp”>
                  <value>”\d”</value>
            </symbol>
      </item>
      <symbol interpretation=”ASCII” value=”,”/>
```

```
</item>
```

Here the 'matchingBehaviour' of the Symbol was set to 'regexp' which means the value of the Symbol to be interpreted as a regular expression. A full definition of the regular expression dialect we currently support is given within the QT documentation.[15]

6.5 HowTo do Addressing

The XCEL puts the emphasis on sequential descriptions. Within sequential descriptions the start position within the **inputfile** is usually calculated by the **XCEL Processor** which means that the start position is defined by the length of the bytstream already consumed by the description matched so far. The example format below shows a string that contains three words, all with length four.

Format:

```
Lovefearread
```

Description:

```
<item identifier=”date” xsi:type=”structuringItem” order=”sequence” >
      <symbol identifier=”oneWord” interpretation=”ASCII”
            length=”4” name=”fourLetterWord” multiple=”true”/>
</item>
```

The description shows how to extract the single words from the format. Given we only want to extract the word fear from the middle of the string. In such cases we can define the start position explicitly.

Description:

```
<item identifier=”date” xsi:type=”structuringItem” order=”sequence” >
<symbol identifier=”oneWord” interpretation=”ASCII” start=”4” length=”4”
      name=”fourLetterWord” />
</item>
```

The latter description extracts only the word fear by giving the concrete start position of the word. The explicit definition of the start position of a given element is not used very oftenly. A more common case is the determination of an explicit start position at runtime; please refer to the “HowTo random access” section to see the details.
An alternative approach to implement addresses within a file is to define a reserved sequence, called separator, to distinguish between records or meaningful parts. Separators can be handled in different ways using the XCEL. One example was given in the “HowTo define constraints”-section with a comma separated list. In the next example we want to show an other possibility which can shorten the description of complex formats significantly. The format we want to describe consist again of three words, separated by two colons.

Format:

```
love::fear::read::
```

Description:

```
<processing type="configureParser">
      <processingMethod name="setAddressingScheme">
            <param value="separator"/>
          <param value="58 58"/>
     </processingMethod>
</processing>
<symbol identifier=”oneWord” interpretation=”ASCII” length=”1”
      name=”fourLetterWord” multiple=”true”/>
```

The description above shows the usage of a **Processing** element to configure the parser. The method 'setAddressingSchema' currently supports two types of addressing 'separator' and 'byte-

[15] Please see <http://doc.trolltech.com/4.2/index.html>

wise'. While the default behaviour provides a byte-wise or more exact a character- or token-wise addressing, with the 'processing' element at the beginning of this example we change the default behaviour and define addresses to be defined by the decimal byte values 58 58 (numerical value for ASCII '::'). The next processing initializes reading by positioning to the first address which starts by definition at file offset zero. The following Symbol is used to read as often as it matches the content starting at one valid address, an occurrence of the string '::' that is. In this example, as in the previous, the Result Tree will contain three Symbols with the values 'love', 'fear', 'read'.
The separator addressing schema does also support the processing methods 'goToFirstAddress', 'goToNextAdress', 'goToPrevAddress' and 'goToLastAddress'.
A typical construction to define a start position within a file is to explicitly define the offset. The example format starts with a number that refers to the offset where the text “Here I am” begins.

Format:

```
8-------Here I am
```

Description:

```
<symbol interpretation=”ASCII” length=”1” identifier=”offset”/>
<processing type="pushXCEL" xcelRef=”HerIAmItem”>
      <processingMethod name="setStart">
                  <param valueRef="offset"/>
       </processingMethod>
</processing>
<item identifier=”HerIAmItem” ...>
...
</item>
```

With the first Symbol we read the offset. The processing of type 'pushXCEL' is used to manipulate an already existing element within the **XCEL Tree**. The element that is to be manipulated is referred by its Id “HerIAmItem”. The processing calls a 'setStart' method and refers in its first parameter to the value of the offset-Symbol we already have read. The next Item which is supposed to contain the proper description of the string will now be applied to byte position eight.
With the given examples we have covered the most important addressing methods. The next section will show how to select a certain address from an offset table.

6.6 HowTo support random access

In this section a frequently used feature called offset table is introduced. An offset table is a structure that stores offsets to certain elements of a file. Usually offset tables are used to enhance the performance for certain operations like read, write or update. PDF is an example for a format that makes heavy use of offset tables.
While the example in this section is less complex than the PDF offsets, it nevertheless provides an overview on how to access certain items within the table to use them as offsets. The example format shows a structure identified by the initial string 'Table:' that holds a list of comma separated numbers terminated by a white space. The numbers within the list can be interpreted as offsets to objects somewhere in the file. To keep it short the objects in the example are simple strings.

Format:

```
Table: 16,24,32 Object1 Object2 Object3
```

Description:

```
<symbol identifier=”TableIdentifier” interpretation=”ASCII” value=”Table:
      ”/>
<item identifier=”TableEntry” xsi:type=”structuringItem” multiple=”true”>
      <symbol identifier=”offset” interpretation=”ASCII” length=”2”/>
      <symbol identifier=”offsetSeparator” interpretation=”ASCII”
            value=”,”/>
</item>
<symbol identifier="ObjectOffset" interpretation="ASCII" length=”0”/>
```

```
<processing type="pushXCEL" xcelRef="ObjectOffset">
     <processingMethod name="getItemNumber">
         <param value="offset"/>
         <param value="2"/>
      </processingMethod>
</processing>
<processing type="pushXCEL" xcelRef="secondObject">
     <processingMethod name="setStartPosition">
          <param valueRef="ObjectOffset"/>
     </processingMethod>
</processing>
```

The snippet above gives an example on how a list of elements read before can be accessed. The **Symbol** at the beginning reads the table and the colon. With the first **Item** we read through the list of offsets at the beginning of our example format. The subsequent Symbol has a length of zero and does therefore not consume any bytes. The Symbol is simply used to reserve an identifier for storing a value (similar to the declaration of a variable in a programming language). The Processing right behind the empty Symbol is now used to ask the processor for all elements with the identifier offset. The second parameter is used to select an element from the list of offsets by its index. The value of the returned element will be assigned to the empty Symbol which is referenced in the 'xcelRef' attribute of the processing. To access the file at the determined position we introduce an other Processing to interpret the value of the 'ObjectOffset'-Symbol as start position of the item that is supposed to process the object. For that the processing refers to the Item and calls the 'setStartposition' method. The first parameter is given a 'valueRef' attribute. While the 'value' attribute requires a number, the 'valueRef' attribute requires an ID of to element read previously whose value is then interpreted as a number. The last Item is a place holder for the description of the referenced object which is not included here.

6.7 HowTo Recursive format descriptions – Encoding Chains

Complex formats often require a single byte stream to be interpreted more than one time. A typical example is the interpretation of a distinct structure after applying a decompression method to it. With the following example we describe how an already processed sequence can be processed again by an **XCEL Tree**. The format is a part of a PDF file that uses so called streams to describe the appearance of graphical objects. PDF streams are usually compressed. To interpret such a stream it must first be uncompressed. The XCEL provides a method to append filter methods to its elements. An element that is read passes its values through the filter chain before the normal process of interpreting the encoding starts. In the example below we assume that the length of the compressed stream is already known.

Format:

```
stream
x□¼u□®□»
□Â0#□¥□÷@□Þ□á□Ì#1I□ÿ□«□¥□â □´{!□à#x##□Á.□¾□¾
i□ª□«□¼□í;□§□Ã□ç□ñ□¶□æ##□ï#□¸g□×□á@□Á□õX□®8□ï□ð\□Ë□¼□ån□Í)Y□Ã□â#□ª
1□ï□Ó#□û) D□¤##□²#□Ò□Ã□°□Ò□û□Ò
Endstream
```

Description:

```
<!-- setLength of streamContent -->
<processing type="pushXCEL" xcelRef="streamContent">
      <processingMethod name="addFilter">
            <param value="compression"/>
                  <param valueRef="Filter"/>
      </processingMethod>
</processing>
<symbol identifier="streamContent" interpretation="ASCII"/>
<item identifier="streamProcessing"  xsi:type="structuringItem"
internalSource="streamContent" >
...
</item>
```

The **Processing** at the beginning refers to the subsequently following symbol which is supposed to read the compressed content of the PDF stream. The comment at the beginning indicates that we assume that the 'streamContent'-**Symbol**'s length has been set previously by some part of the XCEL document not included here. The effect of the processing is, that, when the **XCEL Processor** processes the Symbol the attached filtering method is applied to the extracted byte sequence. The result is a Symbol that holds the decompressed stream as its value. What we need is a possibility to access the value of this symbol and interpret it with another **XCEL Tree**. This feature was been introduced by the 'internalSource' attribute that can be applied to an XCEL **Item**. With the attribute we order the XCEL Processor to proceed with processing the value of the referenced element and not at the original byte stream. Within the Item we can now define a complete XCEL Tree that interprets the content of the uncompressed Symbol.

6.8 HowTo handle Multiple File Formats

The Digital Object paradigm places an emphasize on logical units rather than files. This implies that a Digital Object can be manifest as a single file, as part of a single file or as multiple files or parts of multiple files. Current file formats like DOCX or ODF do actually describe a structure of a set of files, not an individual one. The XCEL has to handle those formats as well; for that it provides tools to access files and parts of files. The example given in this section should provide a demonstration of the navigation between two files. The format is given in the table below were each column describes the content of one file. The first 'main.exp' file consists of three lines where the second line must be interpreted as a path to another file. The content of the second file should be integrated into a unified result tree.

Format:

Main.exp
```
Hello
world.exp
Hello Friends
```

world.exp
```
World
```

Description:
```
<!--read the first line -->
<symbol identifier="WorldFileName" interpretation="ASCII" length="1"/>
<processing type="pushXCEL" xcelRef="WorldFile" >
      <processingMethod name="setExternalSource">
            <param valueRef="WorldFileName"/>
      </processingMethod>
</processing>
<item xsi:type="structuringItem" identifier="WorldFile">
...
</item>
```
The code block above outlines the navigation between two files. Lets assume that the parser is set to read lines not bytes and that the parser's read position is now at the beginning of the second line, so that the first **Symbol** specified above matches the second line of the example format. The subsequent **Processing** is used to set the 'externalSource' attribute of the references 'WorldFile'-Item to the value represented by the 'WorldFileName'-Symbol. So we simply add the information where to continue to the following Item.

6.9 HowTo wrap Tiffinfo

Many existing characterisation tools have been developed to produce formatted textual descriptions of the characteristics of a file. As far as XCEL is concerned, there is no significant difference between such a formatted output and the format of a file. It is also immediately apparent, that by applying an XCEL Processor to the output of an existing characterisation tool, we can increase the number of formats that can be converted into XCDL rapidly.
In the following section we illustrate this by describing how to write an XCEL document for the Tiffinfo output format.

Tiffinfo[16] is a tool that displays information about files created according to the Tagged Image File Format. By default, the contents of each TIFF directory in each file are displayed, with the value of each tag shown symbolically. You can use the Tiffinfo tool with the following options:

-D	read data
-i	ignore read errors
-c	display data for grey/color response curve or colormap
-d	display raw/decoded image data
-f lsb2msb	force lsb-to-msb FillOrder for input
-f msb2lsb	force msb-to-lsb FillOrder for input
-j	show JPEG tables
-o offset	set initial directory offset
-r	read/display raw image data instead of decoded data
-s	display strip offsets and byte counts
-w	display raw data in words rather than bytes
-z	enable strip chopping
-#	set initial directory (first directory is # 0)

The Tiffinfo example we will use is an output of the Tiffinfo tool with the following options -D -c -d -r for the image basi0g08.tif. Each record in this output consists of key value pairs such as 'Image Width: 32' or 'Image Length: 32' describing this image. What we would like to do next is to write an XCEL description for this format.

Figure 8: basi0g08.tif

```
TIFF Directory at offset 0x8 (8)
  Subfile Type: (0 = 0x0)
  Image Width: 32 Image Length: 32
  Resolution: 72, 72 pixels/inch
  Bits/Sample: 8
  Compression Scheme: None
  Photometric Interpretation: min-is-black
  Samples/Pixel: 1
  Rows/Strip: 32
  Planar Configuration: single image plane
  Photoshop Data: <present>, 1298 bytes
  Tag 37724:
0x41,0x64,0x6f,0x62,0x65,0x20,0x50,0x68,0x6f,0x74,0x6f,0x73,0x68,0x6f,0x7
0,0x20,0x44,0x6f,0x63,0x75,0x6d,0x65,0x6e,0x74,0x20,0x44,0x61,0x74,0x61,0
x20,0x42,0x6c,0x6f,0x63,0x6b,0x0
Strip 0:
  00 01 02 03 04 05 06 07 08 09 0a 0b 0c 0d 0e 0f 10 11 12 13 14 15 16 17
  18 19 1a 1b 1c 1d 1e 1f 20 21 22 23 24 25 26 27 28 29 2a 2b 2c 2d 2e 2f
  30 31 32 33 34 35 36 37 38 39 3a 3b 3c 3d 3e 3f 40 41 42 43 44 45 46 47
  48 49 4a 4b 4c 4d 4e 4f 50 51 52 53 54 55 56 57 58 59 5a 5b 5c 5d 5e 5f
  60 61 62 63 64 65 66 67 68 69 6a 6b 6c 6d 6e 6f 70 71 72 73 74 75 76 77
  78 79 7a 7b 7c 7d 7e 7f 80 81 82 83 84 85 86 87 88 89 8a 8b 8c 8d 8e 8f
  90 91 92 93 94 95 96 97 98 99 9a 9b 9c 9d 9e 9f a0 a1 a2 a3 a4 a5 a6 a7
  a8 a9 aa ab ac ad ae af b0 b1 b2 b3 b4 b5 b6 b7 b8 b9 ba bb bc bd be bf
  c0 c1 c2 c3 c4 c5 c6 c7 c8 c9 ca cb cc cd ce cf d0 d1 d2 d3 d4 d5 d6 d7
  d8 d9 da db dc dd de df e0 e1 e2 e3 e4 e5 e6 e7 e8 e9 ea eb ec ed ee ef
  f0 f1 f2 f3 f4 f5 f6 f7 f8 f9 fa fb fc fd fe ff fe fd fc fb fa f9 f8 f7
  f6 f5 f4 f3 f2 f1 f0 ef ee ed ec eb ea e9 e8 e7 e6 e5 e4 e3 e2 e1 e0 df
  de dd dc db da d9 d8 d7 d6 d5 d4 d3 d2 d1 d0 cf ce cd cc cb ca c9 c8 c7
  c6 c5 c4 c3 c2 c1 c0 bf be bd bc bb ba b9 b8 b7 b6 b5 b4 b3 b2 b1 b0 af
```

[16] LIBTIFF, Version 3.8.2 , Copyright (c) 1988-1996 Sam Leffler , Copyright (c) 1991-1996 Silicon Graphics, Inc. , http://www.remotesensing.org/libtiff/

```
  ae ad ac ab aa a9 a8 a7 a6 a5 a4 a3 a2 a1 a0 9f 9e 9d 9c 9b 9a 99 98 97
  96 95 94 93 92 91 90 8f 8e 8d 8c 8b 8a 89 88 87 86 85 84 83 82 81 80 7f
  7e 7d 7c 7b 7a 79 78 77 76 75 74 73 72 71 70 6f 6e 6d 6c 6b 6a 69 68 67
  66 65 64 63 62 61 60 5f 5e 5d 5c 5b 5a 59 58 57 56 55 54 53 52 51 50 4f
  4e 4d 4c 4b 4a 49 48 47 46 45 44 43 42 41 40 3f 3e 3d 3c 3b 3a 39 38 37
  36 35 34 33 32 31 30 2f 2e 2d 2c 2b 2a 29 28 27 26 25 24 23 22 21 20 1f
  1e 1d 1c 1b 1a 19 18 17 16 15 14 13 12 11 10 0f 0e 0d 0c 0b 0a 09 08 07
  06 05 04 03 02 01 00 01 02 03 04 05 06 07 08 09 0a 0b 0c 0d 0e 0f 10 11
  12 13 14 15 16 17 18 19 1a 1b 1c 1d 1e 1f 20 21 22 23 24 25 26 27 28 29
  2a 2b 2c 2d 2e 2f 30 31 32 33 34 35 36 37 38 39 3a 3b 3c 3d 3e 3f 40 41
  42 43 44 45 46 47 48 49 4a 4b 4c 4d 4e 4f 50 51 52 53 54 55 56 57 58 59
  5a 5b 5c 5d 5e 5f 60 61 62 63 64 65 66 67 68 69 6a 6b 6c 6d 6e 6f 70 71
  72 73 74 75 76 77 78 79 7a 7b 7c 7d 7e 7f 80 81 82 83 84 85 86 87 88 89
  8a 8b 8c 8d 8e 8f 90 91 92 93 94 95 96 97 98 99 9a 9b 9c 9d 9e 9f a0 a1
  a2 a3 a4 a5 a6 a7 a8 a9 aa ab ac ad ae af b0 b1 b2 b3 b4 b5 b6 b7 b8 b9
  ba bb bc bd be bf c0 c1 c2 c3 c4 c5 c6 c7 c8 c9 ca cb cc cd ce cf d0 d1
  d2 d3 d4 d5 d6 d7 d8 d9 da db dc dd de df e0 e1 e2 e3 e4 e5 e6 e7 e8 e9
  ea eb ec ed ee ef f0 f1 f2 f3 f4 f5 f6 f7 f8 f9 fa fb fc fd fe ff fe fd
  fc fb fa f9 f8 f7 f6 f5 f4 f3 f2 f1 f0 ef ee ed ec eb ea e9 e8 e7 e6 e5
  e4 e3 e2 e1 e0 df de dd dc db da d9 d8 d7 d6 d5 d4 d3 d2 d1 d0 cf ce cd
  cc cb ca c9 c8 c7 c6 c5 c4 c3 c2 c1 c0 bf be bd bc bb ba b9 b8 b7 b6 b5
  b4 b3 b2 b1 b0 af ae ad ac ab aa a9 a8 a7 a6 a5 a4 a3 a2 a1 a0 9f 9e 9d
  9c 9b 9a 99 98 97 96 95 94 93 92 91 90 8f 8e 8d 8c 8b 8a 89 88 87 86 85
  84 83 82 81 80 7f 7e 7d 7c 7b 7a 79 78 77 76 75 74 73 72 71 70 6f 6e 6d
  6c 6b 6a 69 68 67 66 65 64 63 62 61 60 5f 5e 5d 5c 5b 5a 59 58 57 56 55
  54 53 52 51 50 4f 4e 4d 4c 4b 4a 49 48 47 46 45 44 43 42 41 40 3f 3e 3d
  3c 3b 3a 39 38 37 36 35 34 33 32 31 30 2f 2e 2d 2c 2b 2a 29 28 27 26 25
  24 23 22 21 20 1f 1e 1d 1c 1b 1a 19 18 17 16 15 14 13 12 11 10 0f 0e 0d
  0c 0b 0a 09 08 07 06 05 04 03 02 01 00 01 02 03
```

Figure 9: The output of 'Tiffinfo -D -c -d -r basi0g08.tif'

6.9.1 Step 1 (main structure)

First we have to write down the basic XCEL tags. To validate the XCEL we use our XCELImageTIFFINFOExtension schema. The main structure of XCEL is shown in Figure 10.

```
<?xml version="1.0" encoding="UTF-8"?>
<XCELDocument
xmlns="http://www.planets-project.eu/xcl/schemas/xcl"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.planets-project.eu/xcl/schemas/xcl
schemas/XCELImageTIFFINFOExtension.xsd">
      <preProcessing></preProcessing>
      <formatDescription></formatDescription>
      <templates></templates>
      <postProcessing></postProcessing>
</XCELDocument>
```

Figure 10: The main structure of XCEL

6.9.2 Step 2 (modelling the basic structure)

To model the basic structure of XCEL for any format we extend the <preProcessing> and <formatDescription> tags. See Figure 11.
The processing method 'setSymbolDoesNotMatchHandle'[17] in the tag <preProcessing> with parameter 'skipWhiteSpace' skips the white space, so that the white space does not need to be explicitly treated in XCEL. This is an auxiliary feature of preprocessing.
The XCEL elements describing the Tiffinfo output we put into the tag <formatDescription>.
First we define the structuring items with 'identifier=rootNode' and 'identifier=allRecords multiple=true order=choice'. The structuring items have the responsibility of structuring the elements in the XCEL file. The attribute identifier assigns the unique title to an XCEL element. The

[17] The processing method is described in detail in chapter 2.3.12

value of this attribute can be an arbitrary character string. The attribute 'multiple=true' defines that the processing of this item with all its own children happens repeatedly until one or all of the children elements are matched. The attribute 'order=choice'[18] specifies that only one of the child elements must be matched by this item.
Below you will see the Item 'allRecords' contains all the XCEL elements describing all the key value pairs in the Tiffinfo output file.
Now we begin with description of the XCEL structure processing the Tiffinfo output line by line. The result of line by line processing is stored in an XCEL element as symbol[19] with identifier 'default_Symbol'. This symbol defines that all parsed values apart from CR and LF should be valid characters for this symbol. The attribute 'multiple=true' expresses that a symbol can have more than one character. The attribute 'length=1' defines that the bytes of this element are read character by character.
Following we create more items as children of an 'allRecords' item, one item for every key value pair describing the image properties. You can put definitions of one item with its child elements into the template part also.

```
<?xml version="1.0" encoding="UTF-8"?>
<XCELDocument xmlns="http://www.planets-project.eu/xcl/schemas/xcl"
              xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
              xsi:schemaLocation="http://www.planets -
              project.eu/xcl/schemas/xcl
               schemas/XCELImageExtension.xsd">
      <preProcessing>
            <item xsi:type="structuringItem" identifier="preproc1">
                  <processing type="configureParser">
                        <processingMethod
                              name="setSymbolDoesNotMatchHandle">
                              <param value="skipWhiteSpace"/>
                        </processingMethod>
                  </processing>
            </item>
      </preProcessing>
      <formatDescription>
            <item xsi:type="structuringItem" identifier="rootNode"
                  order="sequence">
                  <item xsi:type="structuringItem"
                        identifier="allRecords" multiple="true"
                        order="choice">
                        <item identifier="default"
                              xsi:type="structuringItem">
                              <symbol identifier="default_Symbol"
                                    length="1" multiple="true"
                                    interpretation="ASCII"
                                    name="uninterpreted">
                                    <nonValidValues>
                                          <value>CR</value>
                                          <value>LF</value>
                                    </nonValidValues>
                              </symbol>
                        </item>
                  </item>
            </item>
      </formatDescription>
      <templates></templates>
      <postProcessing></postProcessing>
</XCELDocument>
```

Figure 11: Modelling the basic structure

[18] The attribute 'order' can have following values: choice, all and sequence. See chapter 2.1.4.3.
[19] The particular definition of symbol is in chapter 2.2.

6.9.3 Step 3 How do we describe a key value pair

In this chapter we will explain how to write an XCEL structure for one specific key value pair from Figure 2.
We decide to describe and to interpret the key value pair of Figure 9 in the third line:

```
Image Width: 32 Image Length: 32
```

The XCEL structure in Figure 12 shows how to describe the key value pair of image property width in XCEL. To do this we define a structuring item 'identifier=imageWidth' containing more child elements like symbol and item.
The symbol 'identifier=imageWidth_Symbol' defines that the row of the Tiffinfo output describing the image property width has to begin with the string 'Image Width:'. Because of that we choose an element of the type symbol with the attributes 'length', 'interpretation' and 'value'. The attribute 'length' determines that 12 bytes should be read at the current parsing position. These 12 bytes should be interpreted in ASCII. This symbol matches, if these 12 bytes contain the string 'Image Width:'.
Next we describe the value that follows the characters 'Image Width:', i.e., the number 32. We ignore the space character which follows 'Image Width:'. This is handled by the method 'skipWhiteSpace'. We define an item of type 'definitionItem' which is an item that adds a name to its children. The following 'imageWidthNumber_Symbol' defines that all ASCII values in range between 0 und 9 should be valid characters for the value of 'imageWidth'. We set the attribute 'multiple=true' to express that a symbol can have more than one character. The attribute 'length =1' defines that the value is read byte by byte. The attributes 'length=1' and 'multiple=true' act like a loop in C++ or Java. It is executed character by character until a invalid value is encountered. The value extracted by this process can be accessed by the name 'imageWidth'[20]. This XCEL structure of Figure 12 must be placed in the tag <formatDescription> as child element of the structuring item with 'identifier=allRecords'. Notice that the item 'identifier=imageWidth_item' comes before the item identifier="default". The reason is that these items have a parent element with attributes 'multiple=true' and 'order=choice' defining that only one of the child elements must be matched by this item. The processing of this element ends if one child element is matched and the following elements are therefore ignored.

```
<item identifier="imageWidth_item" xsi:type="structuringItem">
      <symbol identifier="imageWidth_Symbol" length="12"
            interpretation="ASCII" value="Image Width:"/>
      <item identifier="imageWidthNumber" xsi:type="definitionItem"
            name="imageWidth">
            <symbol identifier="imageWidthNumber_Symbol" length="1"
                  multiple="true" interpretation="ASCII" >
                  <validValues>
                        <startRange>0</startRange>
                        <endRange>9</endRange>
                  </validValues>
            </symbol>
      </item>
</item>
```

Figure 12: The XCEL structure to describing a key value pair 'Image Width: 32'

6.9.4 Step 4 (How do we describe a key value pair with value interpretation)

If a series of bytes or tokens is extracted by reading or rather matching, it can be interpreted as a certain value. *How* the value is to be interpreted is defined by the tag <valueInterpretation>. The tag <value> states what should be matched. The value in the tag <valueLabel> specifies how <value> is to be interpreted.

```
<item identifier="CompressionScheme" xsi:type="structuringItem">
      <symbol identifier="CompressionScheme_Symbol" value="Compression
            Scheme:" length="19" interpretation="ASCII"/>
      <symbol identifier="SP" value="32" optional="true"/>
```

[20] The 'imageWidth' is a name of image property defined by XCELImageTIFFINFOExtension.

```
      <item identifier="CompressionSchemeValue" xsi:type="definitionItem
            name="compression">
            <symbol identifier="CompressionSchemeValue_Symbol" length="4"
                  interpretation="ASCII">
                  <valueInterpretation>
                        <valueLabel>uncompressed</valueLabel>
                        <value>None</value>
                  </valueInterpretation>
            </symbol>
      </item>
</item>
```

Figure 13: The XCEL structure to describing a key value pair with value interpretation

6.9.5 Step 5 (XCEL for the Tiffinfo file format)

Next we quote the full file of XCEL for Tiffinfo format for the sake of completeness. This XCEL is used by running the Extractor tool for the Tiffinfo output by 'tiffinfo -D -c -d -r'.

```
<?xml version="1.0" encoding="UTF-8"?>
<XCELDocument xmlns="http://www.planets-project.eu/xcl/schemas/xcl"
              xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
              xsi:schemaLocation="http://www.planets-
project.eu/xcl/schemas/xcl
              schemas/XCELImageTIFFINFOExtension.xsd">
    <preProcessing>
        <item xsi:type="structuringItem" identifier="preproc1">
            <processing type="configureParser">
                <processingMethod name="setSymbolDoesNotMatchHandle">
                    <param value="skipWhiteSpace"/>
                </processingMethod>
            </processing>
        </item>
    </preProcessing>
    <formatDescription>
        <item xsi:type="structuringItem" identifier="rootNode">
            <item xsi:type="structuringItem" identifier="allRecords"
                  multiple="true" order="choice">

                <!-- imageWidth -->
                <item identifier="imageWidth" xsi:type="structuringItem">
                    <symbol identifier="imageWidth_Symbol" length="12"
                            interpretation="ASCII" value="Image Width:"/>
                    <item identifier="imageWidthNumber" name="imageWidth"
                          xsi:type="definitionItem">
                        <symbol identifier="imageWidthNumber_Symbol"
                              length="1" multiple="true"
                              interpretation="ASCII">
                            <validValues>
                                <startRange>0</startRange>
                                <endRange>9</endRange>
                            </validValues>
                        </symbol>
                    </item>
                </item>

                <!-- imageLength -->
                <item identifier="ImageLength"
```

```
            xsi:type="structuringItem">
        <symbol identifier="ImageLength_Symbol" length="13"
            interpretation="ASCII" value="ImageLength:"/>
        <item identifier="ImageLengthNumber"
            name="imageHeight"
            xsi:type="definitionItem">
            <symbol identifier="ImageLengthNumber_Symbol"
                  length="1"
                  multiple="true" interpretation="ASCII">
                <validValues>
                    <startRange>0</startRange>
                    <endRange>9</endRange>
                </validValues>
            </symbol>
        </item>
    </item>

    <!-- Resolution -->
    <item identifier="Resolution" xsi:type="structuringItem">
        <symbol identifier="Resolution_Symbol" length="11"
                interpretation="ASCII" value="Resolution:"/>
        <item identifier="ResolutionX" name="resolutionX"
              xsi:type="definitionItem">
            <symbol identifier="ResolutionX_Symbol"
                  length="1"
                  multiple="true" interpretation="ASCII">
                <validValues>
                    <startRange>0</startRange>
                    <endRange>9</endRange>
                </validValues>
            </symbol>
        </item>
        <symbol identifier="comma" length="1" value="44"/>
        <symbol identifier="space" length="1" value="32"/>
        <item identifier="ResolutionY" name="resolutionY"
              xsi:type="definitionItem">
              <symbol identifier="ResolutionY_Symbol"
                  length="1"
                    multiple="true" interpretation="ASCII">
                <validValues>
                    <startRange>0</startRange>
                    <endRange>9</endRange>
                </validValues>
            </symbol>
        </item>
        <symbol identifier="pixels_Symbol" length="1"
                multiple="true" interpretation="ASCII">
            <nonValidValues>
                <value>CR</value>
                <value>LF</value>
                <value>/</value>
            </nonValidValues>
        </symbol>
        <symbol identifier="slash" value="/" length="1"
                interpretation="ASCII"></symbol>
        <item identifier="ResolutionUnit"
            name="resolutionUnit"
              xsi:type="definitionItem">
            <symbol identifier="ResolutionUnit_Symbol"
                  length="1"
                    multiple="true" interpretation="ASCII">
                <nonValidValues>
                    <value>CR</value>
```

```
                <value>LF</value>
                <value>/</value>
            </nonValidValues>
        </symbol>
    </item>
</item>

<!--   Bits/Sample:  -->
<item identifier="BitsSample" xsi:type="structuringItem">
    <symbol identifier="BitsSample_Symbol"
        multiple="true"
        value="Bits/Sample:" length="12"
        interpretation="ASCII"/>
    <item identifier="BitsSampleNumber"
        name="bitsPerSample"
          xsi:type="definitionItem">
        <symbol identifier="BitsSampleNumber_Symbol"
              length="1"
                multiple="true" interpretation="ASCII">
            <validValues>
                <startRange>0</startRange>
                <endRange>9</endRange>
            </validValues>
        </symbol>
    </item>
</item>

<!--   Compression Scheme: -->
<item identifier="CompressionScheme"
        xsi:type="structuringItem">
    <symbol identifier="CompressionScheme_Symbol"
        value="Compression Scheme:" length="19"
        interpretation="ASCII"/>
    <symbol identifier="SP" value="32" optional="true"/>
    <item identifier="CompressionSchemeValue"
        name="compression"
          xsi:type="definitionItem">
        <symbol
              identifier="CompressionSchemeValue_Symbol"
              interpretation="ASCII"
              length="4">
            <valueInterpretation>
                <valueLabel>uncompressed</valueLabel>
                <value>None</value>
            </valueInterpretation>
        </symbol>
    </item>
</item>

<!--Photometric Interpretation:-->
<item identifier="PhotometricInterpretation"
      xsi:type="structuringItem">
    <symbol identifier="PhotometricInterpretation_Symbol"
        length="27" interpretation="ASCII"
        value="Photometric Interpretation:"/>
    <symbol identifier="SP2" value="32" optional="true"/>
    <item identifier="PhotometricInterpretationValue"
          name="imageType" xsi:type="definitionItem">
        <symbol

              identifier="PhotometricInterpretationValue_
              Symbol" interpretation="ASCII"
               length="12">
```

```
            <valueInterpretation>
                <valueLabel>blackIsZero</valueLabel>
                <value>min-is-black</value>
            </valueInterpretation>
        </symbol>
    </item>
</item>

<!--   Samples/Pixel:  -->
<item identifier="SamplesPixel"
        xsi:type="structuringItem">
    <symbol identifier="SamplesPixel_Symbol"
        multiple="true"
        value="Samples/Pixel:" length="14"
        interpretation="ASCII"/>
    <item identifier="SamplesPixelValue"
          xsi:type="definitionItem">
        <symbol identifier="SamplesPixelValue_Symbol"
              name="samplesPerPixel" length="1"
              multiple="true" interpretation="ASCII">
            <validValues>
                <startRange>0</startRange>
                <endRange>9</endRange>
            </validValues>
        </symbol>
    </item>
</item>

<!--  Rows/Strip:-->
<item identifier="RowsStrip" xsi:type="structuringItem">
    <symbol identifier="RowsStrip_Symbol" multiple="true"
        value="Rows/Strip:" length="11"
        interpretation="ASCII"/>
     <item identifier="RowsStripValue"
              xsi:type="definitionItem">
        <symbol identifier="RowsStripValue_Symbol"
              name="rowsPerStrip" length="1"
              multiple="true" interpretation="ASCII">
            <validValues>
                <startRange>0</startRange>
                <endRange>9</endRange>
            </validValues>
        </symbol>
    </item>
</item>

<!--  Planar Configuration: -->
<item identifier="PlanarConfiguration"
      xsi:type="structuringItem">
    <symbol identifier="PlanarConfiguration_Symbol"
            multiple="true" value="Planar Configuration:"
            length="21"
            interpretation="ASCII"/>
    <item identifier="PlanarConfigurationValue"
          name="planarConfiguration"
          xsi:type="definitionItem">
        <symbol
            identifier="PlanarConfigurationValue_Symbol"
              length="1" multiple="true"
              interpretation="ASCII">
            <nonValidValues>
                <value>CR</value>
                <value>LF</value>
```

```
                        </nonValidValues>
                    </symbol>
                </item>
            </item>

            <!--  Strip 0:-->
            <item identifier="Strip0" xsi:type="structuringItem">
                <symbol identifier="Strip0_Symbol" value="Strip 0:"
                        length="8" interpretation="ASCII"/>
                <item identifier="Strip0Value"
                    xsi:type="definitionItem"
                      multiple="true">
                    <symbol identifier="Strip0Value_Symbol"
                          name="normData"
                          length="2" multiple="true"
                          interpretation="ASCII">
                        <nonValidValues>
                            <value>CR</value>
                            <value>LF</value>
                            <value>SP</value>
                        </nonValidValues>
                    </symbol>
                </item>
            </item>

            <item identifier="default" xsi:type="structuringItem">
                <symbol identifier="default_Symbol" length="1"
                        multiple="true" interpretation="ASCII">
                    <nonValidValues>
                        <value>CR</value>
                        <value>LF</value>
                    </nonValidValues>
                </symbol>
            </item>

        </item>
    </item>
</formatDescription>
<postProcessing></postProcessing>
</XCELDocument>
```

Figure 14: The XCEL for the Tiffinfo file format

6.10 HowTo wrap Image Magick

The same principle which we have shown with the tiffinfo characterisation tool can be applied to the output of the "identify" tool of the widely used Image Magick image processing system. ImageMagick[21] is a software system designed to create, edit, and compose bitmap images. It can read, convert and write images in a variety of formats (over 100) including DPX, EXR, GIF, JPEG, JPEG-2000, PDF, PhotoCD, PNG, Postscript, SVG, and TIFF. ImageMagick can be used to translate, flip, mirror, rotate, scale, shear and transform images, adjust image colors, apply various special effects, or draw text, lines, polygons, ellipses and Bézier curves.

The identify program is part of the ImageMagick tools. It describes the format and characteristics of one or more image files. It also reports if an image is incomplete or corrupt. The information returned includes the file name, the width and height of the image, whether the image is colour-mapped or not, the number of colours in the image, the number of bytes in the image, the format of the image (JPEG, PNM, etc.), and finally the number of seconds it took to read and process the image. Many more attributes are available with the verbose option[22].

[21] http://www.imagemagick.org/script/index.php
[22] http://www.imagemagick.org/script/identify.php#options

The identify -verbose output looks like the output of the Tiffinfo tool, which has been discussed in the previous chapter 'HowTo wrap Tiffinfo'. It is a list of key value pairs describing the image. The output of the identify -verbose tool for the image basi0g08.tif looks as follows.

```
Image: tiffsuit/basi0g08.tif
  Format: TIFF (Tagged Image File Format)
  Class: DirectClass
  Geometry: 32x32+0+0
  Resolution: 72x72
  Print size: 0.444444x0.444444
  Units: PixelsPerInch
  Type: Grayscale
  Endianess: MSB
  Colorspace: Gray
  Depth: 8-bit
  Channel depth:
    gray: 8-bit
  Channel statistics:
    gray:
      min: 0 (0)
      max: 255 (1)
      mean: 127.008 (0.49807)
      standard deviation: 73.8886 (0.289759)
  Histogram:
         5: (  1,  1,  1) #010101 rgb(1,1,1)
         5: (  2,  2,  2) #020202 rgb(2,2,2)
         5: (  3,  3,  3) #030303 grey1
         4: (  4,  4,  4) #040404 rgb(4,4,4)
         4: (  5,  5,  5) #050505 grey2
         4: (  6,  6,  6) #060606 rgb(6,6,6)
            ...
            ...
         3: (  0,  0,  0) #000000 black
         2: (255,255,255) #FFFFFF white
  Rendering intent: Undefined
  Interlace: None
  Background color: white
  Border color: rgb(223,223,223)
  Matte color: grey74
  Transparent color: black
  Page geometry: 32x32+0+0
  Dispose: Undefined
  Iterations: 0
  Compression: None
  Orientation: TopLeft
  Properties:
    create-date: 2008-04-28T16:41:33+02:00
    modify-date: 2008-04-28T16:41:29+02:00
    signature:
bb0105fe0f0e88ee1bfb570deef6471c8850391a46c4455e341c4345a6ab42d9
    tiff:rows-per-strip: 32
  Profiles:
    Profile-8bim: 1298 bytes
    unknown[77,111]: chrome Halftone Settings
    Profile-tiff:37724: 36 bytes
  Artifacts:
    verbose: true
  Tainted: False
  Filesize: 2.51172kb
  Number pixels: 1kb
  Version: ImageMagick 6.4.0 04/07/08 Q16 http://www.imagemagick.org
```

Figure 15: The output of 'identify -verbose basi0g08.tif'

6.10.1 Step 1 (modelling the basic structure)

We start by describing the basic XCEL structure for reading the output of the ImageMagick tool identify -verbose. We use the processing Method 'skipWhiteSpace' to suppress the white space.

```
<?xml version="1.0" encoding="UTF-8"?>
<XCELDocument
      xmlns="http://www.planets-project.eu/xcl/schemas/xcl"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="http://www.planets-project.eu/xcl/schemas/xcl
      schemas/XCELImageIMAGEMAGICKExtension.xsd">
      <preProcessing>
            <item xsi:type="structuringItem" identifier="preproc1">
                  <processing type="configureParser" >
                  <processingMethod name="setSymbolDoesNotMatchHandle">
                              <param value="skipWhiteSpace"/>
                        </processingMethod>
                  </processing>
            </item>
      </preProcessing>
      <formatDescription>
            <item xsi:type="structuringItem" identifier="rootNode" >
                  <item xsi:type="structuringItem"
                        identifier="allRecords"
                        multiple="true"order="choice">
                  ...
                  ...
                  </item>
            </item>
    </formatDescription>
    <postProcessing></postProcessing>
</XCELDocument>
```

Figure 16: Modelling the basic structure

The item 'structuringItem identifier=allRecords' in the basic XCEL structure should enclose all other elements to be parsed. The attribute 'multiple=true' defines that it is tried repeatedly to prove the structures located in this item.

6.10.2 Step 2 (How do we describe a key value pair)

Recognised key value pairs in the ImageMagick output can be described in XCEL similar to the Tiffinfo XCEL example in the previous chapter. A number of the characteristics extracted by ImageMagick, like 'Geometry: 32x32+0+0', look a bit different from those in the Tiffinfo output. ImageMagick defines the image measures by its Unix-style geometry definition[23]. The attributes of geometry are width, height, xoffset and yoffset. In this case we have an image with width and height of 32 units, here inches. Furthermore, we set a target to write an XCEL structure that allows us to read these two numbers, i.e., 32, as image measures like image width and image height.
The XCEL code such image dimensions looks as shown in the following XCEL code in Figure 17. After the example we explain the code line by line.

```
<item identifier="imageGeometry" xsi:type="structuringItem"
      multiple="false" order="sequence" >

      <symbol identifier="imageGeometry_Symbol" length="9"
            interpretation="ASCII" value="Geometry:"/>
```

[23] The geometry specifications of ImageMagick are in the form "<width>x<height>{+-}<xoffset>{+-}<yoffset>" for specifying the size and placement location for an object.
http://www.imagemagick.org/Magick++/Geometry.html
http://www.imagemagick.org/script/command-line-options.php#geometry

```
      <!-- imageWidth -->
      <item identifier="imageWidth" xsi:type="definitionItem"
            name="imageWidth">
            <symbol identifier="imageWidth_Symbol" length="1"
                  multiple="true" interpretation="ASCII" >
                  <validValues>
                        <startRange>0</startRange>
                        <endRange>9</endRange>
                  </validValues>
            </symbol>
      </item>
      <symbol identifier="item_x" length="1" interpretation="ASCII"
            value="x"/>

      <!-- imageHeight -->
      <item identifier="imageHeight" xsi:type="definitionItem"
            name="imageHeight">
            <symbol identifier="imageHeight_Symbol" length="1"
                  multiple="true" interpretation="ASCII" >
                  <validValues>
                        <startRange>0</startRange>
                        <endRange>9</endRange>
                  </validValues>
            </symbol>
      </item>
      <symbol identifier="rest" length="1" multiple="true"
            interpretation="ASCII">
            <nonValidValues>
                  <value>CR</value>
                  <value>LF</value>
            </nonValidValues>
      </symbol>
</item>
```

Figure 17: The XCEL structure for describing the key vale pair 'Geometry: 32x32+0+0'

The structuring item 'identifier=imageGeometry' groups the necessary elements. It contains all elements described in the imageMagick output line 'Geometry: 32x32+0+0 '. The attribute 'multiple=false' of this structuring item makes sure that this item, and the elements included within it, are applied only once. The attribute 'order=sequence' declares that the elements included in this item, such as symbols and items, should be proven sequentially, i.e., iterative, and in order of appearance .
Items for image dimensions are defined here by items with the identifiers 'imageWidth' and 'imageHeight'. The content parsed in these items is accessible through the names of these identifiers.
The structure 'imageWidth_Symbol' with attributes 'length=1' and 'multiple=true' works like a loop. This means that the content matched by this symbol is read step by step until the 'validValues' condition fails. The loop exits if a character not between 0 and 9 is encountered. The termination condition is defined with the tag <validValues> which defines the acceptable values.
With the symbol 'identifier=item_x' one byte is read by attribute 'length=1'. This single byte must match the value specified by attribute 'value'. In this case this character must be 'x'. A similar symbol structure is available for the symbol 'identifier=imageGeometry_Symbol'. Here nine bytes are read by the attribute 'length=9'. This symbol is recognized if the nine characters read agree with the content of the attribute 'value', i.e., 'Geometry:'. Notice that these symbols also have the attribute 'multiple'. This attribute 'multiple' has the default value 'false' and needs not to be stated explicitly. Furthermore, these symbols do not have a direct parent element having the attribute labelled 'name'. The result is that the matched content in these symbols is not stored temporarily. This is not necessary in this case, as the characters extracted here are not mapped into file characteristics, but simply labels for the identification of image properties.
The property image height is parsed just like image width.

6.10.3 Step 3 (modelling of processing method)

The output of imageMagick tool identify -verbose is dependent on image format and image properties. The output of identify -verbose for Figure 19 contains, for example, 'Colormap'.
Figure 18 shows part of the output of identify -verbose for the image file 'ArcTriomphe-cHRM-red-green-swap.png'.

```
Colormap: 253
      0: (  0,  0,  0) #000000 black
      1: ( 56,120, 28) #38781C rgb(56,120,28)
      2: (140,156,124) #8C9C7C rgb(140,156,124)
      3: (216,248,152) #D8F898 rgb(216,248,152)
      4: (204,248,160) #CCF8A0 rgb(204,248,160)
      5: ( 40,120, 12) #28780C rgb(40,120,12)
      ...
      ...
      251: ( 56,128, 28) #38801C rgb(56,128,28)
      252: ( 52,112, 44) #34702C rgb(52,112,44)
```

Figure 18: The cutout of 'identify -verbose ArcTriomphe-cHRM-red-green-swap.png'

Figure 19: ArcTriomphe-cHRM-red-green-swap.png

The following Figure 20 shows the XCEL structure for matching the output shown in Figure 18. The XCEL code should allow us to parse the highlighted 'Colormap' values. The not highlighted items should be ignored.

```
<item identifier="rgbPaletteComplete" xsi:type="structuringItem">
      <symbol identifier="rgbPalette_Symbol" length="9"
            interpretation="ASCII" value="Colormap:"/>
      <symbol identifier="rest4" length="1" multiple="true"
            interpretation="ASCII">
            <nonValidValues>
                  <value>CR</value>
                  <value>LF</value>
            </nonValidValues>
      </symbol>
      <item identifier="rgbPalette2" xsi:type="structuringItem"
            multiple="true">
            <symbol identifier="rgbNum"length="1" multiple="true"
                  interpretation="ASCII">
                  <validValues>
                        <startRange>0</startRange>
                        <endRange>9</endRange>
                  </validValues>
            </symbol>
            <symbol identifier="rgbPNum" length="3"
                  interpretation="ASCII" value=": ("/>
            <item identifier="rgbPalette_value"
                  xsi:type="structuringItem" multiple="true">
                  <item identifier="rgbPalette_value_n"
```

```
                        xsi:type="definitionItem" multiple="true">
                        <symbol identifier="rgbPalette_value_Symbol"
                              length="1" multiple="true"
                              interpretation="ASCII">
                              <validValues>
                                    <startRange>0</startRange>
                                    <endRange>9</endRange>
                              </validValues>
                        </symbol>
                  </item>
                  <symbol identifier="rgbPalette_komma" length="1"
                        interpretation="ASCII" value=","
                        optional="true"/>
            </item>
            <symbol identifier="rest5" length="1" multiple="true"
                  interpretation="ASCII">
                  <nonValidValues>
                        <value>CR</value>
                        <value>LF</value>
                  </nonValidValues>
            </symbol>
      </item>
      <symbol identifier="rgbPaletteX" name="rgbPalette" length="0"
            interpretation="uint8"/>
      <processing type="pushXCEL" xcelRef="rgbPaletteX">
            <processingMethod name="setValue">
                  <param valueRef="rgbPalette_value_n"/>
                  <param value=" "/>
            </processingMethod>
  </processing>
</item>
```

Figure 20: The modelling of processing method

Next we describe this XCEL structure for 'Colormap' key value pair more comprehensively. The item 'identifier=rgbPaletteComplete' is an XCEL structure for reading the lines of the Figure 18 iteratively . The values from the first bracket in each line are stored as values from 'rgbPalette'[24].
The characters '253' are read by the 'symbol identifier='rgbPalette_Symbol'.
The 'item identifier=rgbPalette2' is a structuring item and contains the all XCEL structures for reading the highlighted values in Figure 18. At first the 'Colormap' values are stored in the XCEL element with 'identifier=rgbPalette_value_Symbol'. Further on in the code this item is accessed by the identifier value 'rgbPalette_value_n'.
Tokens such as '0: (' or '3: (' are read by the symbols 'identifier=rgbNum' and 'identifier=rgbPNum'. The space character after the '(' we can ignore because it is skipped by the processing Method 'skipWhiteSpace', see the XCEL basic structure in Figure 16.
The item 'identifier=rgbPalette_value' is used for structuring XCEL elements.
The item 'identifier=rgbPalette_value_n' is a definition item. The content matched in this item is accessible through the identifier value.
The content of 'identifier=rgbPalette_value_n' is, after successful matching, the number values in ASCII type. The XCEL structure <processing> converts this content of ASCII type into unit8 type using the processing method 'setValue' of processing type 'pushXCEL'. The value, which must be converted by the method 'setValue', is defined by the tag <param valueRef>. The attribute 'xcelRef' of tag <processing> defines the element, which contains the result of ' pushXCEL' processing.

[24] The 'rgbPalette' is a name of image property defined by XCELImageExtension schema. It means the same as 'Colormap' of ImageMagick tool.

7. Appendix

7.1 XCEL Schemas

7.1.1 XCELBasicStructure.xsd

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
        xmlns="http://www.planets-project.eu/xcl/schemas/xcl"
        xmlns:tns="http://www.planets-project.eu/xcl/schemas/xcl"
        targetNamespace="http://www.planets-project.eu/xcl/schemas/xcl"
        elementFormDefault="qualified">
   <!-- ************************************************************************ -->
   <!-- ************************ Basic Structure ************************** -->
   <!-- ************************************************************************ -->
   <xs:element name="XCELDocument">
      <xs:annotation>
         <xs:documentation>
            Basic Structure for XML-Instances based on the eXtensible
            Characterization Extraction Language.
         </xs:documentation>
      </xs:annotation>
      <xs:complexType>
         <xs:sequence>
            <xs:element ref="preProcessing" minOccurs="0"/>
            <xs:element ref="formatDescription"/>
            <xs:element ref="templates" minOccurs="0"/>
            <xs:element ref="postProcessing" minOccurs="0"/>
         </xs:sequence>
      </xs:complexType>
   </xs:element>

   <xs:element name="preProcessing">
      <xs:annotation>
         <xs:documentation>
            The preProcessing section is reserved to perform configuring
            tasks that effects the XCELProcessor.
         </xs:documentation>
      </xs:annotation>
      <xs:complexType>
         <xs:sequence>
            <xs:element ref="item" minOccurs="0" maxOccurs="1"/>
         </xs:sequence>
      </xs:complexType>
   </xs:element>

   <xs:element name="formatDescription">
      <xs:annotation>
         <xs:documentation>
            The preProcessing section is reserved to perform configuring
            tasks that effects the XCELProcessor.
         </xs:documentation>
      </xs:annotation>
      <xs:complexType>
         <xs:sequence>
            <xs:element ref="item" minOccurs="0" maxOccurs="1"/>
         </xs:sequence>
      </xs:complexType>
```

```
</xs:element>

<xs:element name="templates">
   <xs:annotation>
      <xs:documentation>
         The preProcessing section is reserved to perform configuring
         tasks that effects the XCELProcessor.
      </xs:documentation>
   </xs:annotation>
   <xs:complexType>
      <xs:sequence>
         <xs:element ref="item" minOccurs="0" maxOccurs="unbounded"/>
      </xs:sequence>
   </xs:complexType>
</xs:element>

<xs:element name="postProcessing">
   <xs:annotation>
      <xs:documentation>
         The preProcessing section is reserved to perform configuring
         tasks that effects the XCELProcessor.
      </xs:documentation>
   </xs:annotation>
   <xs:complexType>
      <xs:sequence>
         <xs:element ref="item" minOccurs="0"/>
      </xs:sequence>
   </xs:complexType>
</xs:element>

<!-- ********************************************************************************-->
<!-- ********************************** END Basic Structure *************-->
<!-- ********************************************************************************-->
<xs:element name="item" type="itemType">
   <xs:annotation>
      <xs:documentation> Technical: Sequence of bytes. Logical: a
         structuring unit, that splits a file into processing units and may have semantic
         meaning. The length of an item is determined by the length of the incldued
         elements. Each item contains at least one sub-item, which is either a symbol,
         and therefore final or another item and as such divisible into smaller units.
      </xs:documentation>
   </xs:annotation>
</xs:element>

<xs:element name="symbol" type="symbolType">
   <xs:annotation>
      <xs:documentation>
         Technical: Sequence of bytes. Logical: Smallest unit for the
         reader to read. It is always characterized by a range, indicating a position and
         length in the file and a series of attributes. Each symbol is of a certain
         symbolType, which can be derived in underlying schemas to attach format-specific
         properties to further describe its value.
      </xs:documentation>
   </xs:annotation>
</xs:element>

<xs:element name="processing" type="processingType">
   <xs:annotation>
      <xs:documentation>
         A processing element contains instructions, which shall be
         interpreted by a xcel parser.
      </xs:documentation>
```

```
    </xs:annotation>
  </xs:element>

  <!-- ********************************************************************** -->
  <!-- ************************************* Basic Types ***************  -->
  <!-- ********************************************************************** -->
  <!-- ********************************** ItemType ************************ -->
  <xs:complexType name="itemType" abstract="true">
    <xs:annotation>
      <xs:documentation>
        a structuringItem groups several symbols or properties
      </xs:documentation>
    </xs:annotation>
    <xs:sequence>
      <xs:element name="range" type="rangeType" minOccurs="0"/>
      <xs:group ref="subItem" maxOccurs="unbounded"/>
    </xs:sequence>
    <xs:attributeGroup ref="commonAttributes"/>
    <xs:attribute name="order" type="orderType" use="optional" default="sequence"/>
  </xs:complexType>

  <!-- ******************************** SubItem Group ********************** -->
  <xs:group name="subItem">
    <xs:choice>
      <xs:element ref="item" minOccurs="0" maxOccurs="unbounded"/>
      <xs:element ref="symbol" minOccurs="0" maxOccurs="unbounded"/>
      <xs:element ref="processing" minOccurs="0" maxOccurs="unbounded"/>
    </xs:choice>
  </xs:group>

  <!-- ******************** Item implementations ************************-->
  <xs:complexType name="structuringItem">
    <xs:annotation>
      <xs:documentation>
        a structuringItem groups several symbols or properties
      </xs:documentation>
    </xs:annotation>
    <xs:complexContent>
      <xs:extension base="itemType"/>
    </xs:complexContent>
  </xs:complexType>

  <xs:complexType name="definitionItem">
    <xs:annotation>
      <xs:documentation>
      </xs:documentation>
    </xs:annotation>
    <xs:complexContent>
      <xs:extension base="itemType">
        <xs:sequence>
          <xs:element ref="tns:nameGroup" minOccurs="0"/>
        </xs:sequence>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>


  <!-- ********************** Symbol Type ******************************-->
  <xs:complexType name="symbolType">
    <xs:sequence>
      <xs:element name="range" type="rangeType" minOccurs="0"/>
      <xs:choice minOccurs="0">
```

```
            <xs:element name="validValues">
               <xs:complexType>
                  <xs:choice>
                     <xs:element ref="tns:valueGroup" maxOccurs="unbounded"/>
                     <xs:sequence>
                        <xs:element name="startRange"/>
                        <xs:element name="endRange"/>
                     </xs:sequence>
                  </xs:choice>
               </xs:complexType>
            </xs:element>

            <xs:element name="nonValidValues">
               <xs:complexType>
                  <xs:choice>
                     <xs:element ref="tns:valueGroup" maxOccurs="unbounded"/>
                     <xs:sequence>
                        <xs:element name="startRange"/>
                        <xs:element name="endRange"/>
                     </xs:sequence>
                  </xs:choice>
               </xs:complexType>
            </xs:element>

            <xs:element name="valueInterpretation">
               <xs:complexType>
                  <xs:sequence maxOccurs="unbounded">
                     <xs:element ref="tns:valueLabelGroup">
                        <xs:annotation>
                           <xs:documentation>
                              The label, the value found at keyValue is associated with. The parser
                              compares the value found for the current symbol  with the one given as
                              keyVaue and choses the appropriate label for the file
                              property.
                           </xs:documentation>
                        </xs:annotation>
                     </xs:element>
                     <xs:element ref="tns:valueGroup">
                        <xs:annotation>
                           <xs:documentation>
                              Encoded file property characteristic,
                              which is interpretated by keyName.
                           </xs:documentation>
                        </xs:annotation>
                     </xs:element>
                  </xs:sequence>
               </xs:complexType>
            </xs:element>

         </xs:choice>
      </xs:sequence>
      <xs:attributeGroup ref="commonAttributes"/>
      <xs:attributeGroup ref="symbolAttributes"/>
   </xs:complexType>

   <!-- ************************ Processing Type *************************** -->
   <xs:complexType name="processingType">
      <xs:annotation>
         <xs:documentation> </xs:documentation>
      </xs:annotation>
```

```
    <xs:sequence>
      <xs:element name="processingMethod" minOccurs="0" maxOccurs="unbounded">
        <xs:complexType>
          <xs:sequence>
            <xs:element name="param" maxOccurs="unbounded" minOccurs="0">
              <xs:complexType>
                <xs:attribute name="valueRef" type="xs:string" use="optional"/>
                <xs:attribute name="value" type="xs:string" use="optional"/>
                <xs:attribute name="listRef" type="xs:IDREF" use="optional"/>
                <xs:attribute name="nameRef" type="xs:string" use="optional"/>
              </xs:complexType>
            </xs:element>
          </xs:sequence>
          <xs:attribute name="name" type="methodType">
            <xs:annotation>
              <xs:documentation>
                Forward declaration! The methodType must to be
                defined in the extension schema. </xs:documentation>
            </xs:annotation>
          </xs:attribute>
        </xs:complexType>
      </xs:element>
    </xs:sequence>

    <xs:attribute name="type" type="processingTypeType"/>

    <xs:attribute name="xcelRef" type="xs:IDREF"/>

    <xs:attribute name="xcelNameRef" type="xs:string"/>

  </xs:complexType>

  <!-- ********************************************************************** -->
  <!-- ********************************** Attribute Groups ****************** -->
  <!-- ********************************************************************** -->

  <!-- ************************** common Attributes ********************** -->

  <xs:attributeGroup name="commonAttributes">

    <xs:attribute name="identifier" type="xs:ID" use="required">
      <xs:annotation>
        <xs:documentation>
          Unique Identifier, first item starting with "ID" the
          followings with a capital "I" followed by a number.
        </xs:documentation>
      </xs:annotation>
    </xs:attribute>

    <xs:attribute name="optional" type="xs:boolean" use="optional" default="false">
      <xs:annotation>
        <xs:documentation>
          an optional item may or may not appear in the processed
          file. As a default every item is required, i.e. optional=false.
        </xs:documentation>
      </xs:annotation>
    </xs:attribute>

    <xs:attribute name="multiple" type="xs:boolean" use="optional" default="false">
      <xs:annotation>
        <xs:documentation>
          In case multiple is true, the symbol can reappear in other
```

```
            parts of the document. The default is false.
          </xs:documentation>
        </xs:annotation>
      </xs:attribute>

      <xs:attribute name="length" type="xs:unsignedInt">
        <xs:annotation>
          <xs:documentation>
          </xs:documentation>
        </xs:annotation>
      </xs:attribute>

      <xs:attribute name="name" type="nameType" use="optional">
        <xs:annotation>
          <xs:documentation>
            Forward declaration! The nameType must to be defined in the
            extension schema.
          </xs:documentation>
        </xs:annotation>
      </xs:attribute>

      <xs:attribute name="encoding" type="interpretationType">
        <xs:annotation>
          <xs:documentation>
            Forward declaration! The interpretationType must to be
            defined in the extension schema.
          </xs:documentation>
        </xs:annotation>
      </xs:attribute>

      <xs:attribute name="print" type="xs:boolean">
        <xs:annotation>
          <xs:documentation>
          </xs:documentation>
        </xs:annotation>
      </xs:attribute>

      <xs:attribute name="internalSource" type="xs:IDREF">
        <xs:annotation>
          <xs:documentation>
          </xs:documentation>
        </xs:annotation>
      </xs:attribute>

      <xs:attribute name="externalSource" type="xs:anyURI">
        <xs:annotation>
          <xs:documentation>
          </xs:documentation>
        </xs:annotation>
      </xs:attribute>

      <xs:attribute name="normDataRelation" type="xs:boolean" default="false">
        <xs:annotation>
          <xs:documentation>
          </xs:documentation>
        </xs:annotation>
      </xs:attribute>

    </xs:attributeGroup>

    <xs:attributeGroup name="symbolAttributes">
```

```
    <xs:attribute name="matchingBehaviour" type="matchingType" use="optional"
        default="string">
      <xs:annotation>
        <xs:documentation>
        </xs:documentation>
      </xs:annotation>
    </xs:attribute>

    <xs:attribute name="interpretation" type="interpretationType" default="uint8">
      <xs:annotation>
        <xs:documentation>
          Forward declaration! The interpretationType must to be
          defined in the extension schema.
        </xs:documentation>
      </xs:annotation>
    </xs:attribute>

    <xs:attribute name="value" />

  </xs:attributeGroup>

  <!-- ********************************************************************** -->
  <!-- ************************************* More Type****************** -->
  <!-- ********************************************************************** -->

  <xs:simpleType name="matchingType">
    <xs:restriction base="xs:string">
      <xs:enumeration value="string">
      </xs:enumeration>
      <xs:enumeration value="regexp">
      </xs:enumeration>
    </xs:restriction>
  </xs:simpleType>

  <xs:simpleType name="orderType">
    <xs:restriction base="xs:string">

      <xs:enumeration value="all">
        <xs:annotation>
          <xs:documentation>
            An Item with order type all tries to match all
            containing expressions to the next position in the byteStream. The processing of
            the Item is finished when no child expression matches. The Item matches to the
            current byteStream if at least one child expression has matched.
          </xs:documentation>
        </xs:annotation>
      </xs:enumeration>
      <xs:enumeration value="sequence">
        <xs:annotation>
          <xs:documentation>
            An Item with order type sequence tries to match all
            child expressions in the given order to the byteStream. The processing of the
            Item is finished if the last child expression was successful associated to the
            byteStream or if the association of one child fails. The Item matches to the
            byteStream if all child expressions were successful associated to the
            byteStream. The order sequence is the default.
          </xs:documentation>
        </xs:annotation>
      </xs:enumeration>
      <xs:enumeration value="choice">
        <xs:annotation>
          <xs:documentation>
```

```
                An item with type choice tries to match one of its child
                expressions to the given byteStream. The processing of the item is finished if
                one or no child expression matches to the byteSream The Item matches to the
                byteStream if exactly one child expression matches.
              </xs:documentation>
            </xs:annotation>
          </xs:enumeration>
        </xs:restriction>
      </xs:simpleType>

      <xs:simpleType name="processingTypeType">
        <xs:restriction base="xs:string">
          <xs:enumeration value="pushXCEL">
            <xs:annotation>
              <xs:documentation>
                pushes the value that results from an interpreted  XCEL
                structure into another part of the xcel syntax description
              </xs:documentation>
            </xs:annotation>
          </xs:enumeration>
          <xs:enumeration value="pullXCEL">
            <xs:annotation>
              <xs:documentation>
                pulls an XCEL structure, referenced by its ID, into the
                parsing tree
              </xs:documentation>
            </xs:annotation>
          </xs:enumeration>
          <xs:enumeration value="configureParser">
            <xs:annotation>
              <xs:documentation>
              </xs:documentation>
            </xs:annotation>
          </xs:enumeration>
        </xs:restriction>
      </xs:simpleType>

      <xs:complexType name="rangeType" >
        <xs:sequence>
          <xs:element name="startposition" type="xs:int" minOccurs="0">
            <xs:annotation>
              <xs:documentation>
              </xs:documentation>
            </xs:annotation>
          </xs:element>
          <xs:element name="length" type="xs:int" minOccurs="0">
            <xs:annotation>
              <xs:documentation>
              </xs:documentation>
            </xs:annotation>
          </xs:element>
        </xs:sequence>
      </xs:complexType>

      <xs:simpleType name="valueTypeType">
        <xs:restriction base="xs:string">
          <xs:enumeration value="value">
            <xs:annotation>
              <xs:documentation>
                the value is given directly.
              </xs:documentation>
            </xs:annotation>
```

```
        </xs:enumeration>
        <xs:enumeration value="valueRef">
          <xs:annotation>
            <xs:documentation>
              the value references another elements value by ID
            </xs:documentation>
          </xs:annotation>
        </xs:enumeration>
        <xs:enumeration value="nameRef">
          <xs:annotation>
            <xs:documentation>
              the value references another elements value by name
              (usually for postprocessing)
            </xs:documentation>
          </xs:annotation>
        </xs:enumeration>
        <xs:enumeration value="mathEx">
          <xs:annotation>
            <xs:documentation>
              the value is a mathmatical expression, which can contain
              values as well as references of values
            </xs:documentation>
          </xs:annotation>
        </xs:enumeration>
      </xs:restriction>
    </xs:simpleType>

    <!-- ********************************************************************** -->
    <!-- ************************* elements for substitution Groups ********  -->
    <!-- ********************************************************************** -->

    <xs:element name="nameGroup" abstract="true"/>
    <xs:element name="valueGroup" abstract="true"/>
    <xs:element name="valueLabelGroup" abstract="true"/>

</xs:schema>
```

7.1.2 XCELBuildInMethods.xsd

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
        xmlns="http://www.planets-project.eu/xcl/schemas/xcl"
        xmlns:tns="http://www.planets-project.eu/xcl/schemas/xcl"
        targetNamespace="http://www.planets-project.eu/xcl/schemas/xcl"
        elementFormDefault="qualified">

    <xs:simpleType name="basicMethodType">
      <xs:restriction base="xs:string">
        <xs:enumeration value="setByteOrder">
          <xs:annotation>
            <xs:documentation>
            </xs:documentation>
          </xs:annotation>
        </xs:enumeration>
        <xs:enumeration value="setName">
          <xs:annotation>
            <xs:documentation>
            </xs:documentation>
          </xs:annotation>
```

```
</xs:enumeration>
<xs:enumeration value="setStartPosition">
   <xs:annotation>
      <xs:documentation>
      </xs:documentation>
   </xs:annotation>
</xs:enumeration>
<xs:enumeration value="setLength">
   <xs:annotation>
      <xs:documentation>
      </xs:documentation>
   </xs:annotation>
</xs:enumeration>
<xs:enumeration value="setInterpretation">
   <xs:annotation>
      <xs:documentation>
      </xs:documentation>
   </xs:annotation>
</xs:enumeration>
<xs:enumeration value="setOptional">
   <xs:annotation>
      <xs:documentation>
         Accepts one parameter with a "name" value set to either
         true or false. Used to set an elements' "optional" attribute.
      </xs:documentation>
   </xs:annotation>
</xs:enumeration>
<xs:enumeration value="setMultiple">
   <xs:annotation>
      <xs:documentation>
         Accepts one parameter with a "name" value set to either
         true or false. Used to set an elements' "multiple" attribute.
      </xs:documentation>
   </xs:annotation>
</xs:enumeration>
<xs:enumeration value="addFilter">
   <xs:annotation>
      <xs:documentation>
      </xs:documentation>
   </xs:annotation>
</xs:enumeration>
<xs:enumeration value="isEqual">
   <xs:annotation>
      <xs:documentation>
      </xs:documentation>
   </xs:annotation>
</xs:enumeration>
<xs:enumeration value="setSymbolDoesNotMatchHandle">
   <xs:annotation>
      <xs:documentation>
      </xs:documentation>
   </xs:annotation>
</xs:enumeration>
<xs:enumeration value="setAddressingScheme">
   <xs:annotation>
      <xs:documentation>
      </xs:documentation>
   </xs:annotation>
</xs:enumeration>
<xs:enumeration value="goToLastAddress">
   <xs:annotation>
      <xs:documentation>
```

```
            </xs:documentation>
        </xs:annotation>
    </xs:enumeration>
    <xs:enumeration value="goToNextAddress">
        <xs:annotation>
            <xs:documentation>
            </xs:documentation>
        </xs:annotation>
    </xs:enumeration>
    <xs:enumeration value="goToPreviousAddress">
        <xs:annotation>
            <xs:documentation>
            </xs:documentation>
        </xs:annotation>
    </xs:enumeration>
    <xs:enumeration value="getItemNumber">
        <xs:annotation>
            <xs:documentation>
            </xs:documentation>
        </xs:annotation>
    </xs:enumeration>
    <xs:enumeration value="setValue">
        <xs:annotation>
            <xs:documentation>
            </xs:documentation>
        </xs:annotation>
    </xs:enumeration>
    <xs:enumeration value="setIdentifier">
        <xs:annotation>
            <xs:documentation>
            </xs:documentation>
        </xs:annotation>
    </xs:enumeration>
    <xs:enumeration value="interpretAsNumber">
        <xs:annotation>
            <xs:documentation>
            </xs:documentation>
        </xs:annotation>
    </xs:enumeration>
    <xs:enumeration value="setExternalSource">
        <xs:annotation>
            <xs:documentation>
            </xs:documentation>
        </xs:annotation>
    </xs:enumeration>
    <xs:enumeration value="setInternalSource">
        <xs:annotation>
            <xs:documentation>
            </xs:documentation>
        </xs:annotation>
    </xs:enumeration>
    <xs:enumeration value="extract">
        <xs:annotation>
            <xs:documentation>
            </xs:documentation>
        </xs:annotation>
    </xs:enumeration>
    <xs:enumeration value="unzip">
        <xs:annotation>
            <xs:documentation>
            </xs:documentation>
        </xs:annotation>
```

```
        </xs:enumeration>
        <xs:enumeration value="addRelation">
          <xs:annotation>
            <xs:documentation>
            </xs:documentation>
          </xs:annotation>
        </xs:enumeration>
      </xs:restriction>
   </xs:simpleType>

</xs:schema>
```

7.1.3 XCLBasicDataTypesLib.xsd

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- extensible characterisation language: data types library for XCDL and XCEL
    documents -->
<!-- created by PLANETS, PC2, University of Cologne, HKI -->
<!-- version 1.0, October 31, 2006 -->
<!-- change history: no changes -->
<xs:schema
        xmlns="http://www.planets-project.eu/xcl/schemas/xcl"
        xmlns:xs="http://www.w3.org/2001/XMLSchema"
        xmlns:tns="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">

   <!-- ******************* data type definitions ************************ -->
   <xs:simpleType name="xclDataTypeDefinition">
      <xs:union memberTypes="tns:xclDataTypes tns:xclCharacterEncodingTypes"/>
   </xs:simpleType>
   <!-- ********************  data types ******************************** -->
   <xs:simpleType name="xclDataTypes">
      <xs:restriction base="xs:string">
        <xs:enumeration value="decimal">
          <xs:annotation>
            <xs:documentation>
              standard mathematical concept of decimal numbers,
              negativ or positive, no limit on places and decimal places
            </xs:documentation>
          </xs:annotation>
        </xs:enumeration>
        <xs:enumeration value="int">
          <xs:annotation>
            <xs:documentation>
              standard mathematical concept of numbers, negative or
              positive, no limit on places
            </xs:documentation>
          </xs:annotation>
        </xs:enumeration>
        <xs:enumeration value="int8">
          <xs:annotation>
            <xs:documentation>
              signed 8 bits integer
            </xs:documentation>
          </xs:annotation>
```

```
</xs:enumeration>
<xs:enumeration value="int16">
   <xs:annotation>
      <xs:documentation>
         signed 16 bits integer
      </xs:documentation>
   </xs:annotation>
</xs:enumeration>
<xs:enumeration value="int32">
   <xs:annotation>
      <xs:documentation>
         signed 32 bits integer
      </xs:documentation>
   </xs:annotation>
</xs:enumeration>
<xs:enumeration value="int64">
   <xs:annotation>
      <xs:documentation>
         signed 64 bits integer
      </xs:documentation>
   </xs:annotation>
</xs:enumeration>
<xs:enumeration value="uint8">
   <xs:annotation>
      <xs:documentation>
         unsigned 8 bits integer
      </xs:documentation>
   </xs:annotation>
</xs:enumeration>
<xs:enumeration value="uint16">
   <xs:annotation>
      <xs:documentation>
         unsigned 16 bits integer
      </xs:documentation>
   </xs:annotation>
</xs:enumeration>
<xs:enumeration value="uint32">
   <xs:annotation>
      <xs:documentation>
         unsigned 32 bits integer
      </xs:documentation>
   </xs:annotation>
</xs:enumeration>
<xs:enumeration value="uint32Rational">
   <xs:annotation>
      <xs:documentation>
         two 16Bit integers where the first... see TIFF spec
      </xs:documentation>
   </xs:annotation>
</xs:enumeration>
<xs:enumeration value="uint64">
   <xs:annotation>
      <xs:documentation>
         unsigned 64 bits integer
      </xs:documentation>
   </xs:annotation>
</xs:enumeration>
<xs:enumeration value="bin">
   <xs:annotation>
      <xs:documentation>
         binary data.
      </xs:documentation>
```

```
            </xs:annotation>
         </xs:enumeration>
         <xs:enumeration value="string">
            <xs:annotation>
               <xs:documentation>
                  Any encoded character string
               </xs:documentation>
            </xs:annotation>
         </xs:enumeration>
         <xs:enumeration value="XCLLabel">
            <xs:annotation>
               <xs:documentation>
                  a string representing a unique label that conforms to
                  XCL properties ontology
               </xs:documentation>
            </xs:annotation>
         </xs:enumeration>
         <xs:enumeration value="timeISO8601">
            <xs:annotation>
               <xs:documentation>
                  type for full date and time according to ISO 8601
               </xs:documentation>
            </xs:annotation>
         </xs:enumeration>
         <xs:enumeration value="bit">
            <xs:annotation>
               <xs:documentation>
                  smallest information unit representing either 0 or 1
               </xs:documentation>
            </xs:annotation>
         </xs:enumeration>
         <xs:enumeration value="byte">
            <xs:annotation>
               <xs:documentation>
                  sequence of eight bits
               </xs:documentation>
            </xs:annotation>
         </xs:enumeration>
      </xs:restriction>
   </xs:simpleType>
   <xs:simpleType name="xclCharacterEncodingTypes">
      <xs:restriction base="xs:string">
         <xs:enumeration value="Latin1">
            <xs:annotation>
               <xs:documentation>
                  ISO 8859-1 (Latin-1) Characters List
               </xs:documentation>
            </xs:annotation>
         </xs:enumeration>
         <xs:enumeration value="ASCII">
            <xs:annotation>
               <xs:documentation>
                  ASCII (American Standard Code for Information
                  Interchange), is a character encoding based on the English alphabet.
               </xs:documentation>
            </xs:annotation>
         </xs:enumeration>
         <xs:enumeration value="utf-8">
            <xs:annotation>
               <xs:documentation>
                  UTF-8 stands for Unicode Transformation Format-8. It is
                  an octet (8-bit) lossless encoding of Unicode characters
```

```
            </xs:documentation>
          </xs:annotation>
        </xs:enumeration>
        <xs:enumeration value="utf-16">
          <xs:annotation>
            <xs:documentation>
              UTF-16 stands for Unicode Transformation Format-16. It
              is an octet (16-bit) lossless encoding of Unicode characters
            </xs:documentation>
          </xs:annotation>
        </xs:enumeration>
        <xs:enumeration value="iso-646">
          <xs:annotation>
            <xs:documentation>
              UTF-16 stands for Unicode Transformation Format-16. It
              is an octet (16-bit) lossless encoding of Unicode characters
            </xs:documentation>
          </xs:annotation>
        </xs:enumeration>
      </xs:restriction>
    </xs:simpleType>
</xs:schema>
```

7.1.4 XCELBasicExtension.xsd

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
        xmlns="http://www.planets-project.eu/xcl/schemas/xcl"
        xmlns:tns="http://www.planets-project.eu/xcl/schemas/xcl"
        targetNamespace="http://www.planets-project.eu/xcl/schemas/xcl"
        elementFormDefault="qualified">

    <xs:annotation>
      <xs:documentation>
        This is a sample schema for extending the XCEL. Copy the code to a
        file with the name XCEL[DataType]Extension.xsd and include your
        XCEL[DataType]NamesLib.xsd and extend the schema types below with
        your types ( [DataType] refers to a domain of file formats (e.g. Image, Video etc.)).
        Remember that changing the interpretationType and the methodType
        needs for  writing of corresponding Plugins for the XCEL Processor.
      </xs:documentation>
    </xs:annotation>
    <xs:include schemaLocation="XCLBasicDataTypesLib.xsd"/>
    <xs:include schemaLocation="XCELBuildInMethods.xsd"/>
    <xs:include schemaLocation="XCELBasicStructure.xsd"/>

    <xs:simpleType name="extendedNameDefinitions">
      <xs:annotation>
        <xs:documentation>
          Please add your own DataTypes as members to the
          extendedNameDefinitions. The XML validator will
          than allow you to use your definitions in the
          name and labeldValue elements of your XCEL
          instance.
        </xs:documentation>
      </xs:annotation>
      <xs:union memberTypes="xs:string"/>
    </xs:simpleType>
```

```
   <xs:simpleType name="extendedValueDefinitions">
      <xs:annotation>
         <xs:documentation>
            Restrict or extend the extendedValueDefinitions to get more
            control over the content of the value elements in the XCEL instance.
         </xs:documentation>
      </xs:annotation>

      <xs:union memberTypes=" xs:string xs:int"/>

   </xs:simpleType>

   <!--******** Attention ****************************************-->
   <!--******** These types must be handled by plugins *******-->
   <!--******** Do not change if don´t want to write code ******-->

   <xs:simpleType name="interpretationType">
      <xs:annotation>
         <xs:documentation>
            The interpretation Type defines wich Datatypes are available for
            symbols and properties.
         </xs:documentation>
      </xs:annotation>
      <xs:union memberTypes="tns:xclDataTypeDefinition"/>
   </xs:simpleType>


   <xs:simpleType name="methodType">
      <xs:annotation>
         <xs:documentation>
            The Method Type is used to define the known methods that can be
            used in the processing element. Methods are used for
            manipulating the XCEL Tree
            or the binary Data, that is described by the XCEL. A Method must
            have a direct corresponding Plugin in the XCEL Processor.
         </xs:documentation>
      </xs:annotation>
      <xs:union memberTypes="tns:basicMethodType"/>
   </xs:simpleType>



   <!--*************** Do not edit*************************-->
   <xs:simpleType name="nameType">
      <xs:annotation>
         <xs:documentation>

         </xs:documentation>
      </xs:annotation>
      <xs:union memberTypes="extendedNameDefinitions"/>
   </xs:simpleType>
   <xs:element name="value" substitutionGroup="tns:valueGroup">
   </xs:element>
   <xs:element name="name" substitutionGroup="tns:nameGroup"
            type="tns:extendedNameDefinitions"/>
   <xs:element name="valueLabel" substitutionGroup="tns:valueLabelGroup"
            type="tns:extendedNameDefinitions"/>
</xs:schema>
```

7.1.5 XCELImagePNGExtension.xsd

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
        xmlns="http://www.planets-project.eu/xcl/schemas/xcl"
        xmlns:tns="http://www.planets-project.eu/xcl/schemas/xcl"
        targetNamespace="http://www.planets-project.eu/xcl/schemas/xcl"
        elementFormDefault="qualified">

   <xs:include schemaLocation="XCLBasicDataTypesLib.xsd"/>
   <xs:include schemaLocation="XCELBuildInMethods.xsd"/>
   <xs:include schemaLocation="XCELBasicStructure.xsd"/>

   <xs:include schemaLocation="fmt_13_png.xsd"/>
   <xs:simpleType name="extendedNameDefinitions">
      <xs:union memberTypes="tns:fmt_13_png"/>
   </xs:simpleType>

   <xs:simpleType name="extendedValueDefinitions">
      <xs:union memberTypes=" xs:string xs:int"/>
   </xs:simpleType>

   <xs:simpleType name="interpretationType">
      <xs:union memberTypes="tns:xclDataTypeDefinition"></xs:union>
   </xs:simpleType>

   <xs:simpleType name="methodType">
      <xs:union memberTypes="tns:basicMethodType"></xs:union>
   </xs:simpleType>

   <xs:simpleType name="nameType">
      <xs:annotation>
         <xs:documentation>

         </xs:documentation>
      </xs:annotation>
      <xs:union memberTypes="extendedNameDefinitions"/>
   </xs:simpleType>
   <xs:element name="value" substitutionGroup="tns:valueGroup"
           type="tns:extendedValueDefinitions"/>
   <xs:element name="name" substitutionGroup="tns:nameGroup"
           type="tns:extendedNameDefinitions"/>
   <xs:element name="valueLabel" substitutionGroup="tns:valueLabelGroup"
           type="xs:string"/>
</xs:schema>
```

7.1.6 fmt_13_png.xsd

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- extensible characterisation language: PNG image  properties' name definitions
      library for XCDL and XCEL documents -->
<!-- created by PLANETS, PC2, University of Cologne (HKI)-->
<!-- version: 09/08/2007 -->
<!-- change history:  -->
<xs:schema
        xmlns:xs="http://www.w3.org/2001/XMLSchema"
        xmlns="http://www.planets-project.eu/xcl/schemas/xcl"
        xmlns:tns="http://www.planets-project.eu/xcl/schemas/xcl"
```

```
        targetNamespace="http://www.planets-project.eu/xcl/schemas/xcl"
        elementFormDefault="qualified">

    <xs:simpleType name="fmt_13_png" id="properties">
      <xs:restriction base="xs:string">
        <xs:enumeration value='signature' id='id4'>
          <xs:annotation>
            <xs:documentation>
              A sequence of bytes that uniquely identifies a certain fileformat
            </xs:documentation>
          </xs:annotation>
        </xs:enumeration>
        <xs:enumeration value='imageWidth' id='id30'>
          <xs:annotation>
            <xs:documentation>
              Width of an image. Corresponds to the x-axis. [Compatibility: PNG 1.1; TIFF 6.0]
            </xs:documentation>
          </xs:annotation>
        </xs:enumeration>
        <xs:enumeration value='imageHeight' id='id2'>
          <xs:annotation>
            <xs:documentation>
              Height of an image. Corresponds to the y-axis of a
              Cartesian coordinate system.. [Compatibility: PNG 1.1; TIFF 6.0]
            </xs:documentation>
          </xs:annotation>
        </xs:enumeration>
        <xs:enumeration value='bitsPerSample' id='id151'>
          <xs:annotation>
            <xs:documentation>
              Number of bits per sample component [Compatibility: PNG 1.1; TIFF 6.0]
            </xs:documentation>
          </xs:annotation>
        </xs:enumeration>
        <xs:enumeration value='imageType' id='id20'>
          <xs:annotation>
            <xs:documentation>
              Interpretation of image pixels representing the colour
              type of an image
            </xs:documentation>
          </xs:annotation>
        </xs:enumeration>
        <xs:enumeration value='compression' id='id18'>
          <xs:annotation>
            <xs:documentation>
              algorithm applied to image data for the purpose of minimizing storage size
            </xs:documentation>
          </xs:annotation>
        </xs:enumeration>
        <xs:enumeration value='filter' id='id39'>
          <xs:annotation>
            <xs:documentation>
              transformation applied to an array of scanlines with the
              aim of improving their compressibility [Compatibility: PNG 1.1]
            </xs:documentation>
          </xs:annotation>
        </xs:enumeration>
        <xs:enumeration value='interlace' id='id45'>
          <xs:annotation>
            <xs:documentation>
              algorithm for encoding raster image data in multiple
              layers with the purpose of improved transmission and
```

```
            image built up [Compatibility: PNG 1.1]
         </xs:documentation>
      </xs:annotation>
   </xs:enumeration>
   <xs:enumeration value='normData' id='id7'>
      <xs:annotation>
         <xs:documentation>
            XCL specific data,
         </xs:documentation>
      </xs:annotation>
   </xs:enumeration>
   <xs:enumeration value='gamma' id='id41'>
      <xs:annotation>
         <xs:documentation>
            numerical parameter used to describe approximations to
            certain non-linear transfer functions encountered in image capture and
            reproduction [Compatibility: PNG 1.1]
         </xs:documentation>
      </xs:annotation>
   </xs:enumeration>
   <xs:enumeration value='rgbPalette' id='id25'>
      <xs:annotation>
         <xs:documentation>
            palette for index-coloured image type. Each pixel is a
            composition of red, green and blue colour values which
            shall appear in exactly this order
         </xs:documentation>
      </xs:annotation>
   </xs:enumeration>
   <xs:enumeration value='significantBits' id='id52'>
      <xs:annotation>
         <xs:documentation>
            number of bits that are significant in the samples [Compatibility: PNG 1.1]
         </xs:documentation>
      </xs:annotation>
   </xs:enumeration>
   <xs:enumeration value='timeLastMod' id='id62'>
      <xs:annotation>
         <xs:documentation>
            last modification date of source object [Compatibility: PNG 1.1]
         </xs:documentation>
      </xs:annotation>
   </xs:enumeration>
   <xs:enumeration value='backgroundColour' id='id32'>
      <xs:annotation>
         <xs:documentation>
            solid colour for the background of an image to be used
            when presenting the image [Compatibility: PNG 1.1]
         </xs:documentation>
      </xs:annotation>
   </xs:enumeration>
   <xs:enumeration value='histogram' id='id42'>
      <xs:annotation>
         <xs:documentation>
            approximate usage frequency of each colour in the
            palette index (rgbPalette) [Compatibility: PNG 1.1]
         </xs:documentation>
      </xs:annotation>
   </xs:enumeration>
   <xs:enumeration value='transparency' id='id27'>
      <xs:annotation>
         <xs:documentation>
```

```
            Alpha information that allows the reference image to be reconstructed
         </xs:documentation>
      </xs:annotation>
   </xs:enumeration>
   <xs:enumeration value='whitePointX' id='id28'>
      <xs:annotation>
         <xs:documentation>
            chromaticity of a computer display's nominal white x value
         </xs:documentation>
      </xs:annotation>
   </xs:enumeration>
   <xs:enumeration value='whitePointY' id='id29'>
      <xs:annotation>
         <xs:documentation>
            chromaticity of a computer display's nominal white y value
         </xs:documentation>
      </xs:annotation>
   </xs:enumeration>
   <xs:enumeration value='1931CIE_ChromaticityRedX' id='id126'>
      <xs:annotation>
         <xs:documentation>
            value x of pair xy specifying red colour according to 1931 CIE color space
         </xs:documentation>
      </xs:annotation>
   </xs:enumeration>
   <xs:enumeration value='1931CIE_ChromaticityRedY' id='id127'>
      <xs:annotation>
         <xs:documentation>
            value y of pair xy specifying red colour according to 1931 CIE color space
         </xs:documentation>
      </xs:annotation>
   </xs:enumeration>
   <xs:enumeration value='1931CIE_ChromaticityGreenX' id='id128'>
      <xs:annotation>
         <xs:documentation>
            value x of pair xy specifying green colour according to 1931 CIE color space
         </xs:documentation>
      </xs:annotation>
   </xs:enumeration>
   <xs:enumeration value='1931CIE_ChromaticityGreenY' id='id129'>
      <xs:annotation>
         <xs:documentation>
            value y of pair xy specifying green colour according to 1931 CIE color space
         </xs:documentation>
      </xs:annotation>
   </xs:enumeration>
   <xs:enumeration value='1931CIE_ChromaticityBlueX' id='id130'>
      <xs:annotation>
         <xs:documentation>
            value x of pair xy specifying blue colour according to
            1931 CIE color space
         </xs:documentation>
      </xs:annotation>
   </xs:enumeration>
   <xs:enumeration value='1931CIE_ChromaticityBlueY' id='id131'>
      <xs:annotation>
         <xs:documentation>
            value y of pair xy specifying green colour according to 1931 CIE color space
         </xs:documentation>
      </xs:annotation>
   </xs:enumeration>
   <xs:enumeration value='textualDataKeyword' id='id58'>
```

```
      <xs:annotation>
        <xs:documentation>
          keyword for textual information associated with the
          image [Compatibility: PNG 1.1]
        </xs:documentation>
      </xs:annotation>
    </xs:enumeration>
    <xs:enumeration value='compressedText' id='id124'>
      <xs:annotation>
        <xs:documentation>
          Character or sequence of characters, without any other
          semantic meaning, but to separate two meaningful units
        </xs:documentation>
      </xs:annotation>
    </xs:enumeration>
    <xs:enumeration value='ICC-1/ICC-1A_RenderingIntent' id='id141'>
      <xs:annotation>
        <xs:documentation>
          suggestion for rendering an image according to
          ICC-1/ICC-1A [Compatibility: PNG 1.1]
        </xs:documentation>
      </xs:annotation>
    </xs:enumeration>
    <xs:enumeration value='resolutionX' id='id23'>
      <xs:annotation>
        <xs:documentation>
          number of pixels per resolution unit in horizontal direction
        </xs:documentation>
      </xs:annotation>
    </xs:enumeration>
    <xs:enumeration value='resolutionY' id='id24'>
      <xs:annotation>
        <xs:documentation>
          number of pixels per resolution unit in vertical direction
        </xs:documentation>
      </xs:annotation>
    </xs:enumeration>
    <xs:enumeration value='resolutionUnit' id='id22'>
      <xs:annotation>
        <xs:documentation>
          measure unit for measuring resolution
        </xs:documentation>
      </xs:annotation>
    </xs:enumeration>
    <xs:enumeration value='suggestedPaletteName' id='id53'>
      <xs:annotation>
        <xs:documentation>
          name of a reduced palette that may be used when the
          display device is not capable of displaying the full
          range of colours in the image [Compatibility: PNG 1.1]
        </xs:documentation>
      </xs:annotation>
    </xs:enumeration>
    <xs:enumeration value='suggestedPaletteSampleDepth' id='id54'>
      <xs:annotation>
        <xs:documentation>
          sample depth of a suggested palette [Compatibility: PNG 1.1]
        </xs:documentation>
      </xs:annotation>
    </xs:enumeration>
    <xs:enumeration value='suggestedPaletteAlpha' id='id56'>
      <xs:annotation>
```

```
            <xs:documentation>
               alpha value for suggested palette [Compatibility: PNG 1.1]
            </xs:documentation>
         </xs:annotation>
      </xs:enumeration>
      <xs:enumeration value='suggestedPaletteFrequency' id='id57'>
         <xs:annotation>
            <xs:documentation>
               frequency of suggested palette  [Compatibility: PNG 1.1]
            </xs:documentation>
         </xs:annotation>
      </xs:enumeration>
      <xs:enumeration value='textualDataString' id='id59'>
         <xs:annotation>
            <xs:documentation>
               Textual information associated with the image
               [Compatibility: PNG 1.1]
            </xs:documentation>
         </xs:annotation>
      </xs:enumeration>
   </xs:restriction>
</xs:simpleType>

</xs:schema>
```

7.1.7 XCELImageTIFFExtension.xsd

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
        xmlns="http://www.planets-project.eu/xcl/schemas/xcl"
        xmlns:tns="http://www.planets-project.eu/xcl/schemas/xcl"
        targetNamespace="http://www.planets-project.eu/xcl/schemas/xcl"
        elementFormDefault="qualified">
   <xs:include schemaLocation="XCLBasicDataTypesLib.xsd"/>
   <xs:include schemaLocation="XCELBuildInMethods.xsd"/>
   <xs:include schemaLocation="XCELBasicStructure.xsd"/>

   <xs:include schemaLocation="fmt_10_tiff.xsd"/>
   <xs:simpleType name="extendedNameDefinitions">
      <xs:union memberTypes="tns:fmt_10_tiff"/>
   </xs:simpleType>

   <xs:simpleType name="extendedValueDefinitions">
      <xs:union memberTypes=" xs:string xs:int"/>
   </xs:simpleType>

   <xs:simpleType name="interpretationType">
      <xs:union memberTypes="tns:xclDataTypeDefinition"></xs:union>
   </xs:simpleType>

   <xs:simpleType name="methodType">
      <xs:union memberTypes="tns:basicMethodType"></xs:union>
   </xs:simpleType>
   <xs:simpleType name="nameType">
      <xs:annotation>
         <xs:documentation>
         </xs:documentation>
      </xs:annotation>
      <xs:union memberTypes="extendedNameDefinitions"/>
```

```
    </xs:simpleType>
    <xs:element name="value" substitutionGroup="tns:valueGroup"
            type="tns:extendedValueDefinitions"/>
    <xs:element name="name" substitutionGroup="tns:nameGroup"
            type="tns:extendedNameDefinitions"/>
    <xs:element name="valueLabel" substitutionGroup="tns:valueLabelGroup" type="
            xs:string"/>
</xs:schema>
```

7.1.8 fmt_10_tiff.xsd

```
<!-- extensible characterisation language: TIFF image properties' name definitions
      library for XCDL and XCEL documents -->
<!-- created by PLANETS, PC2, University of Cologne (HKI )-->
<!-- version: 09/08/2007 -->
<!-- change history:  -->
<xs:schema
        xmlns:xs="http://www.w3.org/2001/XMLSchema"
        xmlns="http://www.planets-project.eu/xcl/schemas/xcl"
        xmlns:tns="http://www.planets-project.eu/xcl/schemas/xcl"
        targetNamespace="http://www.planets-project.eu/xcl/schemas/xcl"
        elementFormDefault="qualified">

    <xs:simpleType name="fmt_10_tiff" id="properties">
      <xs:restriction base="xs:string">
        <xs:enumeration value='byteOrder' id='id6'>
          <xs:annotation>
            <xs:documentation>
              Ordering of bytes for multi-byte data values within a file
            </xs:documentation>
          </xs:annotation>
        </xs:enumeration>
        <xs:enumeration value='signature' id='id4'>
          <xs:annotation>
            <xs:documentation>
              A sequence of bytes that uniquely identifies a certain
              fileformat
            </xs:documentation>
          </xs:annotation>
        </xs:enumeration>
        <xs:enumeration value='byteOffsetRef' id='id117'>
          <xs:annotation>
            <xs:documentation>

            </xs:documentation>
          </xs:annotation>
        </xs:enumeration>
        <xs:enumeration value='dataType' id='id118'>
          <xs:annotation>
            <xs:documentation>

            </xs:documentation>
          </xs:annotation>
        </xs:enumeration>
        <xs:enumeration value='countValues' id='id119'>
          <xs:annotation>
            <xs:documentation>
```

```
        </xs:documentation>
      </xs:annotation>
    </xs:enumeration>
    <xs:enumeration value='newSubfileType' id='id142'>
      <xs:annotation>
        <xs:documentation>
          A general indication of the kind of data contained in
          this subfile [cit. TIFF 6.0] [Compatibility: TIFF 6.0]
        </xs:documentation>
      </xs:annotation>
    </xs:enumeration>
    <xs:enumeration value='subfileType' id='id146'>
      <xs:annotation>
        <xs:documentation>
          A general indication of the kind of data contained in
          this subfile [cit. TIFF 6.0] [Compatibility: TIFF 6.0]
        </xs:documentation>
      </xs:annotation>
    </xs:enumeration>
    <xs:enumeration value='imageWidth' id='id30'>
      <xs:annotation>
        <xs:documentation>
          Width of an image. Corresponds to the x-axis.
          [Compatibility: PNG 1.1; TIFF 6.0]
        </xs:documentation>
      </xs:annotation>
    </xs:enumeration>
    <xs:enumeration value='imageHeight' id='id2'>
      <xs:annotation>
        <xs:documentation>
          Height of an image. Corresponds to the y-axis.
          [Compatibility: PNG 1.1; TIFF 6.0]
        </xs:documentation>
      </xs:annotation>
    </xs:enumeration>
    <xs:enumeration value='bitsPerSample' id='id151'>
      <xs:annotation>
        <xs:documentation>
          Number of bits per sample component [Compatibility: PNG 1.1; TIFF 6.0]
        </xs:documentation>
      </xs:annotation>
    </xs:enumeration>
    <xs:enumeration value='bitsPerSampleRed' id='id12'>
      <xs:annotation>
        <xs:documentation>
          Number of bits per red colour channel [Compatibility:
          PNG 1.1; TIFF 6.0]
        </xs:documentation>
      </xs:annotation>
    </xs:enumeration>
    <xs:enumeration value='bitsPerSampleGreen' id='id14'>
      <xs:annotation>
        <xs:documentation>
          Number of bits per green colour channel [Compatibility: PNG 1.1; TIFF 6.0]
        </xs:documentation>
      </xs:annotation>
    </xs:enumeration>
    <xs:enumeration value='bitsPerSampleBlue' id='id15'>
      <xs:annotation>
        <xs:documentation>
          Number of bits per blue colour channel [Compatibility: PNG 1.1; TIFF 6.0]
        </xs:documentation>
```

```
    </xs:annotation>
  </xs:enumeration>
  <xs:enumeration value='extraSamples' id='id37'>
    <xs:annotation>
      <xs:documentation>
        Description of extra components [cit. TIFF 6.0] [Compatibility: TIFF 6.0]
      </xs:documentation>
    </xs:annotation>
  </xs:enumeration>
  <xs:enumeration value='compression' id='id18'>
    <xs:annotation>
      <xs:documentation>
        algorithm applied to image data for the purpose of minimizing storage size
      </xs:documentation>
    </xs:annotation>
  </xs:enumeration>
  <xs:enumeration value='imageType' id='id20'>
    <xs:annotation>
      <xs:documentation>
        Interpretation of image pixels representing the colour  type of an image
      </xs:documentation>
    </xs:annotation>
  </xs:enumeration>
  <xs:enumeration value='thresholding' id='id61'>
    <xs:annotation>
      <xs:documentation>
        For black and white TIFF files that represent shades of gray, the technique used to
      </xs:documentation>
    </xs:annotation>
  </xs:enumeration>
  <xs:enumeration value='cellWidth' id='id134'>
    <xs:annotation>
      <xs:documentation>
        The width of the dithering or halftoning matrix used to
        create a dithered or halftoned bilevel file (x-axis) [Compatibility: TIFF 6.0]
      </xs:documentation>
    </xs:annotation>
  </xs:enumeration>
  <xs:enumeration value='cellHeight' id='id133'>
    <xs:annotation>
      <xs:documentation>
        The height of the dithering or halftoning matrix used to
        create a dithered or halftoned bilevel file (y-axis)  [Compatibility: TIFF 6.0]
      </xs:documentation>
    </xs:annotation>
  </xs:enumeration>
  <xs:enumeration value='fillOrder' id='id21'>
    <xs:annotation>
      <xs:documentation>
        The logical order of bits within a byte.
      </xs:documentation>
    </xs:annotation>
  </xs:enumeration>
  <xs:enumeration value='scanDocName' id='id144'>
    <xs:annotation>
      <xs:documentation>
        The name of the document from which this image was
        scanned [cit. TIFF 6.0] [Compatibility: TIFF 6.0]
      </xs:documentation>
    </xs:annotation>
  </xs:enumeration>
  <xs:enumeration value='imageDescription' id='id44'>
```

```
      <xs:annotation>
        <xs:documentation>
          A string that describes the subject of the image [cit. TIFF 6.0]
          [Compatibility: TIFF 6.0]
        </xs:documentation>
      </xs:annotation>
    </xs:enumeration>
    <xs:enumeration value='scannerName' id='id51'>
      <xs:annotation>
        <xs:documentation>
          the scanner manufacturer [cit. TIFF 6.0] [Compatibility: TIFF 6.0]
        </xs:documentation>
      </xs:annotation>
    </xs:enumeration>
    <xs:enumeration value='scannerModel' id='id50'>
      <xs:annotation>
        <xs:documentation>
          the scanner model name or number. [cit. TIFF 6.0] [Compatibility: TIFF 6.0]
        </xs:documentation>
      </xs:annotation>
    </xs:enumeration>
    <xs:enumeration value='orientation' id='id48'>
      <xs:annotation>
        <xs:documentation>
          The orientation of the image with respect to the rows
          and columns  [cit. TIFF 6.0] [Compatibility: TIFF 6.0]
        </xs:documentation>
      </xs:annotation>
    </xs:enumeration>
    <xs:enumeration value='samplesPerPixel' id='id26'>
      <xs:annotation>
        <xs:documentation>
          number of intersections (components) of a pixel in an image
        </xs:documentation>
      </xs:annotation>
    </xs:enumeration>
    <xs:enumeration value='rowsPerStrip' id='id116'>
      <xs:annotation>
        <xs:documentation>
          number of rows in each strip except possibly the last
          strip [cit. TIFF 6.0] [Compatibility: TIFF 6.0]
        </xs:documentation>
      </xs:annotation>
    </xs:enumeration>
    <xs:enumeration value='stripByteCount' id='id145'>
      <xs:annotation>
        <xs:documentation>
          For each strip, the number of bytes in the strip after
          compression [cit. TIFF 6.0] [Compatibility: TIFF 6.0]
        </xs:documentation>
      </xs:annotation>
    </xs:enumeration>
    <xs:enumeration value='minSampleValue' id='id46'>
      <xs:annotation>
        <xs:documentation>
          The minimum component value used [cit. TIFF 6.0] [Compatibility: TIFF 6.0]
        </xs:documentation>
      </xs:annotation>
    </xs:enumeration>
    <xs:enumeration value='maxSampleValue' id='id47'>
      <xs:annotation>
        <xs:documentation>
```

```
            The maximum component value used [cit. TIFF 6.0] [Compatibility: TIFF 6.0]
         </xs:documentation>
      </xs:annotation>
   </xs:enumeration>
   <xs:enumeration value='resolutionX' id='id23'>
      <xs:annotation>
         <xs:documentation>
            number of pixels per resolution unit in horizontal direction
         </xs:documentation>
      </xs:annotation>
   </xs:enumeration>
   <xs:enumeration value='resolutionY' id='id24'>
      <xs:annotation>
         <xs:documentation>
            number of pixels per resolution unit in vertical direction
         </xs:documentation>
      </xs:annotation>
   </xs:enumeration>
   <xs:enumeration value='planarConfiguration' id='id49'>
      <xs:annotation>
         <xs:documentation>
            How the components of each pixel are stored [cit. TIFF 6.0]
            [Compatibility: TIFF 6.0]
         </xs:documentation>
      </xs:annotation>
   </xs:enumeration>
   <xs:enumeration value='freeOffsets' id='id137'>
      <xs:annotation>
         <xs:documentation>
            For each string of contiguous unused bytes in a TIFF
            file, the byte offset of the string [cit. TIFF 6.0] [Compatibility: TIFF 6.0]
         </xs:documentation>
      </xs:annotation>
   </xs:enumeration>
   <xs:enumeration value='freeByteCounts' id='id136'>
      <xs:annotation>
         <xs:documentation>
            For each string of contiguous unused bytes in a TIFF
            file, the number of bytes in the string [cit. TIFF 6.0] [Compatibility: TIFF 6.0]
         </xs:documentation>
      </xs:annotation>
   </xs:enumeration>
   <xs:enumeration value='greyResponseUnit' id='id139'>
      <xs:annotation>
         <xs:documentation>
            The precision of the information contained in the
            GrayResponseCurve [cit. TIFF 6.0] [Compatibility: TIFF 6.0]
         </xs:documentation>
      </xs:annotation>
   </xs:enumeration>
   <xs:enumeration value='greyResponseCurve' id='id138'>
      <xs:annotation>
         <xs:documentation>
            For greyscale data, the optical density of each possible
            pixel value [cit. TIFF 6.0] [Compatibility: TIFF 6.0]
         </xs:documentation>
      </xs:annotation>
   </xs:enumeration>
   <xs:enumeration value='resolutionUnit' id='id22'>
      <xs:annotation>
         <xs:documentation>
            measure unit for measuring resolution
```

```
      </xs:documentation>
    </xs:annotation>
  </xs:enumeration>
  <xs:enumeration value='creationSoftware' id='id3'>
    <xs:annotation>
      <xs:documentation>
        Word 2002 and Word 2003 allow the RTF emitter
        application to stamp the document with its name
      </xs:documentation>
    </xs:annotation>
  </xs:enumeration>
  <xs:enumeration value='creationDate' id='id36'>
    <xs:annotation>
      <xs:documentation>
        Date and time of image creation [cit. TIFF 6.0]
        [Compatibility: TIFF 6.0]
      </xs:documentation>
    </xs:annotation>
  </xs:enumeration>
  <xs:enumeration value='artist' id='id31'>
    <xs:annotation>
      <xs:documentation>
        Person who created the image [cit. TIFF 6.0]
        [Compatibility: TIFF 6.0]
      </xs:documentation>
    </xs:annotation>
  </xs:enumeration>
  <xs:enumeration value='hostCompOS' id='id140'>
    <xs:annotation>
      <xs:documentation>
        The computer and/or operating system in use at the time
        of image creation [cit. TIFF 6.0] [Compatibility: TIFF 6.0]
      </xs:documentation>
    </xs:annotation>
  </xs:enumeration>
  <xs:enumeration value='rgbPalette' id='id25'>
    <xs:annotation>
      <xs:documentation>
        palette for index-coloured image type. Each pixel is a
        composition of red, green and blue colour values which
        shall appear in exactly  this order
      </xs:documentation>
    </xs:annotation>
  </xs:enumeration>
  <xs:enumeration value='tileWidth' id='id147'>
    <xs:annotation>
      <xs:documentation>
        The tile width in pixels. This is the number of columns in each tile (322)
      </xs:documentation>
    </xs:annotation>
  </xs:enumeration>
  <xs:enumeration value='tileLength' id='id148'>
    <xs:annotation>
      <xs:documentation>
        The tile length (height) in pixels. This is the number of rows in each tile. (323)
      </xs:documentation>
    </xs:annotation>
  </xs:enumeration>
  <xs:enumeration value='tileByteCounts' id='id150'>
    <xs:annotation>
      <xs:documentation>
        For each tile, the number of (compressed) bytes in that tile. (325)
```

```
        </xs:documentation>
      </xs:annotation>
    </xs:enumeration>
    <xs:enumeration value='extraSampleX' id='id38'>
      <xs:annotation>
        <xs:documentation>
          Description of extra components [cit. TIFF 6.0] [Compatibility: TIFF 6.0]
        </xs:documentation>
      </xs:annotation>
    </xs:enumeration>
    <xs:enumeration value='XMLText' id='id125'>
      <xs:annotation>
        <xs:documentation>
          used in bmp
        </xs:documentation>
      </xs:annotation>
    </xs:enumeration>
    <xs:enumeration value='copyright' id='id35'>
      <xs:annotation>
        <xs:documentation>
          Copyright notice of the person or organization that
          claims the copyright to the image [cit. TIFF 6.0] [Compatibility: TIFF 6.0]
        </xs:documentation>
      </xs:annotation>
    </xs:enumeration>
    <xs:enumeration value='IPTC' id='id65'>
      <xs:annotation>
        <xs:documentation>
          IPTC (International Press Telecommunications Council) metadata (33723)
        </xs:documentation>
      </xs:annotation>
    </xs:enumeration>
    <xs:enumeration value='photoshop' id='id64'>
      <xs:annotation>
        <xs:documentation>
          Used by the GDAL library, holds an XML list of
          name=value 'metadata' values about the image as a whole,
          and about specific samples (42112)
        </xs:documentation>
      </xs:annotation>
    </xs:enumeration>
    <xs:enumeration value='GDALMetadata' id='id63'>
      <xs:annotation>
        <xs:documentation>
          Used by the GDAL library, holds an XML list of
          name=value 'metadata' values about the image as a whole,
          and about specific samples (42112)
        </xs:documentation>
      </xs:annotation>
    </xs:enumeration>
    <xs:enumeration value='normData' id='id7'>
      <xs:annotation>
        <xs:documentation>
          XCL specific data
        </xs:documentation>
      </xs:annotation>
    </xs:enumeration>
    <xs:enumeration value='normDataOffset' id='id143'>
      <xs:annotation>
        <xs:documentation>
          A general indication of the kind of data contained in
          this subfile [cit. TIFF 6.0] [Compatibility: TIFF 6.0]
```

```
            </xs:documentation>
          </xs:annotation>
        </xs:enumeration>
        <xs:enumeration value='tileOffset' id='id149'>
          <xs:annotation>
            <xs:documentation>
              For each tile, the byte offset of that tile, as  compressed and stored on disk (324)
            </xs:documentation>
          </xs:annotation>
        </xs:enumeration>
      </xs:restriction>
    </xs:simpleType>

</xs:schema>
```

8. Reference

1. Sebastian Beyl, Volker Heydegger et. al. Final XCDL Specification, 2008.
2. The DFDL – Data Format Description Language
   http://forge.gridforum.org/sf/projects/dfdl-wg
3. Ecma Office Open XML File Formats Standard
   http://www.ecma-international.org/news/TC45_current_work/TC45_available_docs.htm
4. K.Fisher, R. Gruber: PADS: A Domain-Specific Language for Processing Ad Hoc Data, in: Proceedings of the 2005 ACM SIGPLAN conference on Programming language design and implementation, 2005.
5. K. Fisher, Y. Mandelbaum, D. Walker: The next 700 Data Description Languages,in: Conference record of the 33rd ACM SIGPLAN-SIGACT symposium on Principles of programming languages, 2006.

6. PDF Specification
   http://www.adobe.com/devnet/pdf/pdf_reference.html
7. PNG Specification
   http://www.w3.org/TR/PNG/
8. Pronom Unique Identifiers
   http://www.nationalarchives.gov.uk/aboutapps/pronom/puid.htm
9. TIFF Specification
   http://partners.adobe.com/public/developer/tiff/index.html
10. W3C XML Schema Specification
   http://www.w3.org/XML/Schema