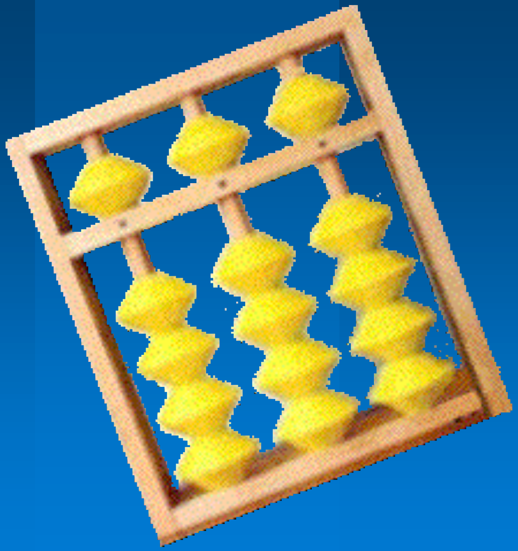
Paris 2004



XML in ABINIT

X. Gonze

Overview

- An introduction to XML (4/5 of the talk)
- In ABINIT :
  - CML I/O
  - Brief description of the XMLf90 lib of Alberto Garcia

Tackling the data interchange problem !
XML + NetCDF
Code reuse !

Why is XML important ?

(Douglas Lovell, IBM T.J. Watson Research Center)

- “For the first time in the history of computing, we have a universally acceptable syntax rich enough to handle all kinds of structured information”
- “XML represents a fundamental change in computing ... away from proprietary file and data formats to a world of open interchange”
- “The driver for this change is the desire by companies and individuals to access and exploit the mass of information made available via the internet”

Goals of this introduction

- To answer the following questions :
  - What is the XML syntax ?
  - What are its advantages over other data representations ?
  - What is a well-formed XML document ?
  - What is a valid XML document ?
  - What are the existing tools and standards to manipulate XML files, especially in view of interchange of data, over the Web ?

Overview

- HTML : the Web markup language
- What is XML ? Rules for a well-formed document
- Defining a markup language :
  - DTDs (Document Type Declarations)
  - XML Schemas
- Climbing the tree structure of XML : XPath
- Programming interfaces : DOM and SAX
- Transformation of a XML document : XSLT
- XLink, XQuery, RDF, SOAP

HTML

- Hyper Text Markup Language

- Pros
  - Easy to use (proliferation of web pages)
  - Hyperlink support, multimedia support
  - Very good industry support for the user
  - Authors write pages displaying information
  - Portability and easy delivery over the network

- Cons
  - A fixed set of tags
  - Content and presentation mixed together

HTML : an example

```
<HTML>   <HEAD>
            <TITLE>Welcome-Readme</TITLE>
           </HEAD>
<BODY>
  <H1>
   <CENTER> <IMG SRC="Images/pcpm.gif" ALIGN=bottom> </CENTER>
  </H1>
<P> <HR>
<p></P> Dear user of ABINIT (in short : ABINITioner),
<p> If this is the first time that you have access to ABINIT,
 or that you receive an ABINIT announcement, welcome !
<p> On the Web site, you will find a lot of things, including installation notes for
 different <a href="http://www.abinit.org/index.html#availables"> versions</a>
 of ABINIT,<a href="http://www.abinit.org/index.html#PSP">pseudopotentials</a>,
 some <a href="http://www.abinit.org/index.html#utile">utilities</a>,
  ....
```

What is XML ?

- XML stands for EXtensible Markup Language
- XML is a “meta-language” to devise markup languages
- XML tags are not predefined in XML. You must define your own tags
- XML syntax is strict
- XML uses a Document Type Definition (DTD) or an XML Schema to formulate a language
- XML with a DTD or XML Schema is designed to be self-descriptive
- Proposed by the W3C (World Wide Web consortium) in 1999
- Ancestor : SGML (1980, already DTDs, but was too complex)

W3C WORLD WIDE WEB consortium

Leading the Web to its Full Potential...

XML Languages

- **XML = Meta-language used to define languages**
- **Examples of languages defined using XML:**

- **MathML** - **Math**ematical **M**arkup **L**anguage
- **XML Schema** - **Schema for XML documents**
- **SVG** - **S**calable **V**ector **G**raphics (a bit like postscript)
- **XSL** - e**X**tensible **S**tyle **L**anguage
- **XHTML** - **X H**yper **T**ext **M**arkup **L**anguage
- **CML** - **C**hemical **M**arkup **L**anguage
- (as of today, hundreds of DTDs available)

A first XML example

`<?xml version="1.0"?>`	Header
`<List_of_participants>`	Root element
`<Organizer id="id1">`	Element with attribute (id)
`<FirstName>Gilles</FirstName>`	
`<LastName>Zerah</LastName>`	Simple elements
`<Language>French</Language>`	
`<Language>English</Language>`	Second occurence of Language
`<Picture url="portrait.gif"/>`	Empty element with attribute (link)
`</Organizer>`	
`</List_of_participants>`	

```xml
<?xml version="1.0"?>
 <List_of_participants>
  <Organizer id="id1">
   <FirstName>Gilles</FirstName>
   <LastName>Zerah</LastName>
   <Language>French</Language>
   <Language>English</Language>
   <Picture url="portrait.gif"/>

  </Organizer>
 </List_of_participants>
```

Well-formed documents

- Each start-tag must have an associated end-tag
- Special markup for empty elements

```
<IMG SRC="picture.gif"/>
equivalent to : <IMG SRC="picture.gif"> </IMG>
```

- Elements must nest properly

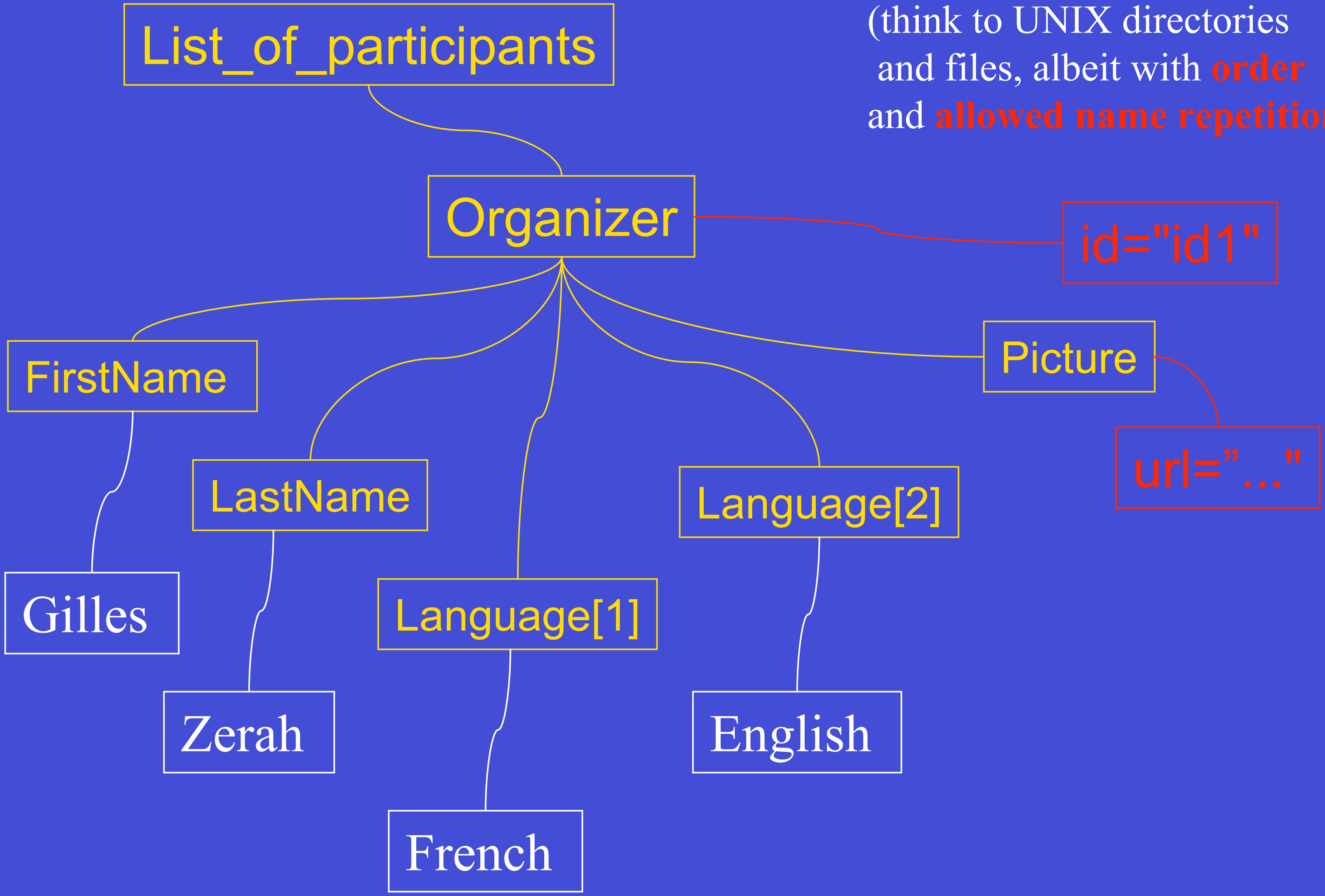
HTML : <b> Haha <i> Hoho </b> Hihi </i>   Wrong in XML

- Documents must have a single root element
- Upper/Lower case matters
- An element cannot have empty attributes

`<DL COMPACT>`	`<DL COMPACT="">`
Wrong	Right

Tree structure of XML

(think to UNIX directories and files, albeit with **order** and **allowed name repetition**)



Valid documents

- A **well-formed** document does not have any constraint about type of elements, attributes ..., and their content, but it fulfills the basic rules of XML
- A **valid** document must be a well-formed document **and** must comply with a grammar

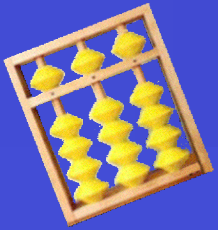
  (allowed elements, attributes ...)

- One mechanism for specifying a grammar is called a **DTD**, another relies on a **XML Schema**

DTD

- Document Type Definition
- Set of syntactic rules for a type of document (Grammar definition language)
- A document can be validated against a DTD
  (xmllint is a simple validator on UNIX/Linux platforms)
- Grammar of a DTD file is NOT XML structured

- Definition of possible elements and their content
- Definition of possible attributes
- + ... (see later)



DTDs : definition of elements

- *Syntax : <!ELEMENT name content>*

- *Examples :*

Simple content

```
<!ELEMENT FirstName (#PCDATA) >
<!ELEMENT LastName (#PCDATA) >
<!ELEMENT Language (#PCDATA) >    PCDATA="parsed character data"
<!ELEMENT Picture EMPTY >
```

Complex content

```
<!ELEMENT Organizer
   (FirstName,LastName,(Language)*,Picture?) >
<!ELEMENT List_of_participants (Organizer|Speaker)* >
```

`* = 0,1 or more , ? = 0 or 1 , + = 1 or more , | = "or" , ","="and"`

DTDs : definition of attributes

- *Syntax : <!ATTLIST element-name*
  (multiple) *attribute-name type default*>

- *Examples :*

```
<!ATTLIST Organizer
               id     ID   #implied >
                      ID="identifier"
<!ATTLIST Picture
               url    CDATA   #required >
                      CDATA="character data"
```

A full DTD

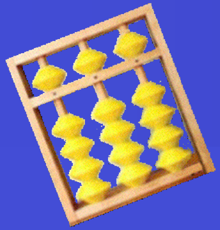
```
<!ELEMENT FirstName (#PCDATA) >
<!ELEMENT LastName (#PCDATA) >
<!ELEMENT Language (#PCDATA) >
<!ELEMENT Picture EMPTY >
<!ELEMENT Organizer
  (FirstName,LastName,(Language)*,Picture?) >
<!ELEMENT Speaker
  (FirstName,LastName,(Language)*,Picture?) >
<!ELEMENT List_of_participants (Instructor|Student)* >
<!ATTLIST Organizer
               id    ID  #implied >
<!ATTLIST Speaker
               id    ID  #implied >
<!ATTLIST Picture
               url   CDATA  #required >
```

Specifying a DTD in a XML file (I)

```
First possibility : no DTD !

<?xml version="1.0"?>
<List_of_participants>
     <Organizer id="id1">
     ….
     </Organizer>
</List_of_participants>
```

A XML parser will be able to check whether the document is well-formed, but it will not check whether it is valid



Specifying a DTD in a XML file (II)

Second possibility : mention the DTD in the document !

```
<?xml version="1.0"?>
<!DOCTYPE List_of_participants [
  <!ELEMENT List_of_participant ... ! Here, one mentions
      ...                           ! the DTD
  <!ATTLIST ...                     !
 ]>
<List_of_participants>
     <Organizer id="id1">
     ....
```

A XML parser will be able to check whether the document is well-formed and whether it is valid. But the DTD would better be independent of the document.

Specifying a DTD in a XML file (III)

Third possibility : reference to the DTD file !

```
<?xml version="1.0"?>
<!DOCTYPE List_of_participants                    !DTD reference
      SYSTEM “List_of_participants.dtd” > !
<List_of_participants>
     <Organizer id="id1">
     ….
```

The List_of_participants.dtd file contains :

```
<!ELEMENT List_of_participant ...
<!ATTLIST ...
```

Problems with the DTD mechanism

- The syntax is specific to the DTD mechanism !

  Not even an XML file ...

  It is contradictory to claim to have a universally acceptable syntax, and not use it to specify the XML languages !
- The DTD typing possibilities are very weak :

  Cannot define an integer, a float, a boolean variable, a date,

  a URL, while grammar rules might be made stronger by relying on such types.

- So, development of new specifications :
  - XML Schema (W3C recommendation, May 2001) Next slides
  - RELAX NG (ISO/IEC technical recommendation)
  - Schematron

A XML Schema is an XML file

A XML file, with a particular grammar !

Also specified by a XML Schema ... of course.

Mechanism : the XML “name space”

```
<?xml version="1.0"?>                                     ! The header
<xs:schema xmlns:xs=“http://www.w3.org/2001/XMLSchema”>
   (Here, one will use elements of the XML Schema
    language, all prefixed by  xs:...  )
</xs:schema>
```

XML Schemas : simple elements

- *Syntax of simple elements (do not have children, do not have attributes):*

```
    <xs:element name="element_name" type="element_type"/>
```

(Note that this syntax is the one of an empty XML element)

- *Examples :*

```
<xs:element name="FirstName" type="xs:string" />
<xs:element name="LastName" type="xs:string" />
<xs:element name="Language" type="xs:string" />
```

Different simple types are possible :

xs:string, xs:ID, xs:anyURI, xs:float, xs:double, xs:integer, xs:boolean, xs:dateTime, ...

(more than 40 simple types)

XML Schemas : complex elements

- *Syntax (complex elements with children, but no attribute):*

```
<xs:element name="..." >
 <xs:complexType>
  <xs:sequence>
    Here, the list of permitted elements, referenced
  </xs:sequence>
 </xs:complexType>
</xs:element>
```

- *List of permitted elements : references, and occurence specification, example :*

```
<xs:element ref="Unique_mandatory_element" />
<xs:element ref="Repeated_element" maxOccurs="unbounded" />
<xs:element ref="Optional_element" minOccurs="1" />
```

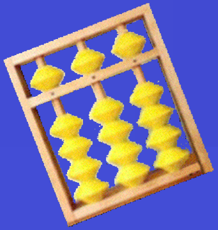
XML Schemas : attributes

- *Syntax of attribute definitions*
  (similar to syntax of element definitions):

```
<xs:attribute name="..." type="..."/>
```

- *Mention an attribute to an element :*

```
<xs:element name="...">
 <xs:complexType>
  <xs:sequence>
   Here, the list of permitted elements, referenced
  </xs:sequence>
  <xs:attribute ref="name_of_attribute"/>    ! HERE
 </xs:complexType>
</xs:element>
```

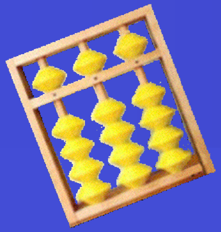


XML Schemas : a full example (I)

- *The XML schema corresponding to the previous DTD*

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
 <xs:element name="FirstName" type="xs:string" />
 <xs:element name="LastName" type="xs:string" />
 <xs:element name="Language" type="xs:string" />
 <xs:attribute name="id" type="xs:ID" />
 <xs:attribute name="url" type="xs:anyURI" />
 <xs:element name="Picture">
  <xs:complexType>
   <xs:attribute ref="url"/>
  </xs:complexType>
 </xs:element>
```

(continued ...)



XML Schemas : a full example (II)

```
<xs:element name="Organizer">
 <xs:complexType>
  <xs:sequence>
   <xs:element ref="FirstName"/>
   <xs:element ref="LastName"/>
   <xs:element ref="Language" minOccurs="0"
                                 maxOccurs="unbounded" >
   <xs:element ref="Picture" minOccurs="0" >
  </xs:sequence>
  <xs:attribute ref="id"/>
 </xs:complexType>
</xs:element>
```

(continued ...)

XML Schemas : a full example (III)

```
<xs:element name="Speaker">
 <xs:complexType>
  <xs:sequence>
   <xs:element ref="FirstName"/>
   <xs:element ref="LastName"/>
   <xs:element ref="Language" minOccurs="0"
                                  maxOccurs="unbounded" >
   <xs:element ref="Picture" minOccurs="0" >
  </xs:sequence>
  <xs:attribute ref="id"/>
 </xs:complexType>
</xs:element>
```

(continued ...)

XML Schemas : a full example (IV)

```
<xs:element name="List_of_participants">
 <xs:complexType>
  <xs:sequence>
   <xs:element ref="Organizer" minOccurs="0"
                                    maxOccurs="unbounded" >
   <xs:element ref="Speaker" minOccurs="0"
                                    maxOccurs="unbounded" >
  </xs:sequence>
 </xs:complexType>
</xs:element>
</xs:schema>                           (the end)
```

XML Schema is much more verbose than the corresponding DTD !

Beyond the language definition

- Suppose that we have a DTD or a XML Schema, and a set of XML documents that are well-formed and valid (can be validated by the DTD or XML schema)
- This rich and well-defined structure allows other layers of standards !
- XPath, API (DOM and SAX), XSLT, XLink, XQuery, RDF, SOAP ... (so many new acronyms)

XPath/XPointer (I)

- A standard to address any part or set of parts of an XML document
- Very similar to UNIX/Linux paths
- Examples of absolute paths :

/	select the root node of the document
/List_of_participants/Instructor	select the "Instructor" node(s), children of the List_of_participants node
/List_of_participants/Instructor/Language[2]	select the second "Language" node, in the specified path
/List_of_participants/Instructor/Language[2]/text()	select the text in the second "Language" node, in the specified path
/List_of_participants/Instructor/@id	select the "id" attribute in the "Instructor" node(s), in the specified path
//Language	select all the "Language" node(s), descendants of the root

XPath (II)

- **Examples of relative paths (need to know the “context node”):**

..	select the parent node
Instructor	select the “Instructor” child(ren), if any

- **Also :**
  - **wild cards ;**
  - **predicates ;**
  - **axis addressing (child,parent,self,attribute,ancestor,descendant, ...)**
  - **functions (count the number of nodes ...) ;**
  - **boolean logic**

API

- Application Programming Interfaces
- Structure of XML document known

⇩

possibility to define

standardized parsing methodologies

(please, do not reinvent the wheel)

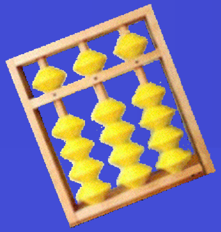
Parsers written in : Python, Perl, C, C++, Java, F90 ...

Two standardized API methodologies :

SAX (Simple API for XML)

DOM (Document Object Model)

SAX



- Simple API for XML
- Idea 1 : Read the XML document sequentially
- Idea 2 : Consider each element, attribute, etc ... , as an "event", that will trigger an "action"
- Idea 3 : SAX routines to be integrated in a language-specific parser, that includes also routines defining the "action" triggered by each event type

- Advantage : the document need not be stored in memory
- However, the on-the-flight treatment of the events is not always easy to code !

DOM

- Document Object Model
- Idea : read the whole XML document, and represent it by a tree in main memory
- Need : the possibility to handle the tree data structure - allocation of pointers (F77 NO, F90 OK)
- The DOM specification is a recommendation of W3C

- Type of objects (all DOM applications use the same names !): Document, Element, Attr, Text ...
- Methods to act on the objects : set(), get() ...
- DOM usually based on SAX !

Transformation of XML documents

- Idea : one wants to automate (and standardize) the generation of .html, .pdf, .ps, .tex, ... or even other XML files from the XML documents

$$\begin{tikzcd}
& & \boxed{\text{XML}} \arrow[dll] \arrow[dl] \arrow[d] \arrow[dr] \arrow[drr] & & \\
\boxed{\text{HTML}} & \boxed{\text{PDF}} & \boxed{\text{PS}} & \boxed{\text{TEX}} & \boxed{\text{XML}}
\end{tikzcd}$$

- Oldest technique : Cascading Style Sheet (formatting)
- New approach (XML based !) :

EXtensible Stylesheet Language for Transformations

Other acronyms

- XLink : XML Linking Language
  - Allows to create (hyper)links between resources (XML documents)
  - Recommendation of the W3C (http://www.w3.org/TR/xlink)
- XQuery : XML Query Language
  - a query language for databases, based on XPath
  - Similarities with SQL
  - http://www.w3.org/XML/Query
- RDF : Resource Description Framework
  - a standard vocabulary to represent Metadata
  - goal : interoperability between applications that exchange informations to be treated automatically (Web oriented)
  - http://www.w3.org/TR/REC-rdf-syntax
- SOAP : Simple Object Access Protocol
  - protocol for exchaning information in a distributed environment
  - http://www.w3.org/TR/SOAP

In ABINIT ...

Use of CML

A CML file :

```
<?xml version="1.0" encoding="iso-8859-1"?>
<molecule id="crystal1" xmlns="http://www.xml-cml.org/schema/cml2/core">
 <crystal>
  <scalar title="a" units="angstrom">10.583544166</scalar>
  <scalar title="b" units="angstrom">10.583544166</scalar>
  <scalar title="c" units="angstrom">15.875316249</scalar>
  <scalar title="alpha" units="degrees">90.000</scalar>
  <scalar title="beta"  units="degrees">90.000</scalar>
  <scalar title="gamma" units="degrees">90.000</scalar>
 </crystal>
 <atomArray>
  <atom id="1" elementType="H" xFract="0.125" yFract="0.000" zFract="0.666666666667"/>
  <atom id="2" elementType="C" xFract="0.250" yFract="0.375" zFract="0.666666666667"/>
  <atom id="3" elementType="O" xFract="0.750" yFract="0.750" zFract="0.500"/>
  <atom id="4" elementType="Si" xFract="0.000" yFract="0.000" zFract="0.000"/>
  </atomArray>
</molecule>
```

Reading a CML file

Input variable keyword :

`cmlfile`

Should be followed by the CML filename string :

(example from Test_v3/t68.in)

```
#   This file is to be complemented by a CML file.
#   Here, only non-CML data are stored.
#   The system will be  : Mo surface 5 layers of 2 atoms + 3 of vacuum

 cmlfile ../t68.in_CML.xml
 diemac 1.0d0
 diemix 0.125d0
 ecut 5.5
```

Reading a CML file (2)

Will initialize :

acell, angdeg, ntypat, natom, typat, xred (xcart)

Might also initialize (if present) :

nsym, symrel, tnons

Note : Superceded by information present in the usual input file. Like a layer of initialisation between the default values and the actual values in the input file.

Routines : append_cml2.f, with parent importcml.f and children findmarkup.f, getattribute.f

Idea : to append the CML file, properly treated, to the ABINIT input file string.

BUT THIS PARSER IS VERY PRIMITIVE !

Printing a CML file

Input variable keyword :

`prtcml`

Should be followed by a positive integer ...

Will write, in a CML file, crystal parameters, symmetry operations,then, for each atom, its number, its type, and its reduced coordinates.

THIS OUTPUT IS OK !

Might be read by other software accepting CML2 syntax !

Other possible future XML usages

Pseudopotential files :

should be produced by pseudopotential generators, then read by ABINIT ...

Might solve the problem of pseudopotential files formats ? ... Only if different softwares (to generate psps, and then, to read psps) agree on the specifications of a XML language ...

OTHER OUTPUT FILES ?

Might be also produced using NetCDF ?

Open discussion ...

The XMLf90 library

Present XML parsing capabilities of ABINIT are too weak !

One year ago, there were no standard XML parser in F90.

Due to the FSAtom action, A. Garcia has developed a library called XMLf90, for parsing XML from FORTRAN 90 codes ...

XMLf90-1.1 is present in ABINITv4.3

Location : ~ABINIT/Lib_XMLf90/xmlf90-1.1.tar

In ~ABINIT, issue

```
make xmlf90
```

This produces a directory xmlf90-1.1, with different subdirectories :

Examples	LICENSE	ReleaseNotes-1.1	macros	xpath
KNOWN_ISSUES	README	Tutorial	sax	

Still have to compile ... (integration is less advanced than NetCDF)

SAX methodology + XPATH methodology

Summary

- A brief introduction to XML
- Status of CML in ABINIT
- The XMLf90 library