XML in FORTRAN

Alberto García
Universidad del País Vasco, Bilbao, SPAIN

XML in Fortran?

- Aren't there nicer alternatives in Python, Java, C++ ... ?
  - If your code is in Fortran you might not want to take the extra step.
  - Who said that Fortran is not nice?
- Can't you just wrap some C parser routines in Fortran?
  - No if you care about portability.

- Native parser in Fortran90
- SAX interface
- Higher-level XPath-like interface
- Portable and reasonably fast
- Small memory footprint

SAX paradigm

```
<item id="003">
  <description>Washing machine</description>
  <price currency="euro">1500.00</price>
</item>
```

Events:

Begin element: item
Begin element: description
PCDATA chunk: “Washing machine”
End element: description
Begin element: price
PCDATA chunk: “1500.00”
End element: price
End element: item

SAX API in Fortran

```
program simple
use flib_sax

type(xml_t)           :: fxml      ! XML file object (opaque)
integer               :: iostat    ! Return code (0 if OK)

call open_xmlfile("inventory.xml",fxml,iostat)
if (iostat /= 0) stop "cannot open xml file"

call xml_parse(fxml, begin_element_handler=begin_element_print)

end program simple
```

Event handlers

```
subroutine begin_element_print(name,attributes)
   character(len=*), intent(in)     :: name
   type(dictionary_t), intent(in)   :: attributes

   character(len=3)  :: id
   integer           :: status

   print *, "Start of element: ", name
   if (has_key(attributes,"id")) then
     call get_value(attributes,"id",id,status)
     print *, "  Id attribute: ", id
   endif
end subroutine begin_element_print
```

Attribute dictionary (Python-like)

```
<person name="john" age="35" />

attributes: { name : john ;     key:value
              age  : 35   }

has_key(attributes,"name")      (function)
get_value(attributes,"name",value,status)

number_of_entries(attributes)  (function)
print_dict(attributes)
...
```

```xml
<inventory>
      <item id="003">
         <description>Washing machine</description>
         <price currency="euro">1500.00</price>
      </item>
      <item id="007">
         <description>Microwave oven</description>
         <price currency="euro">300.00</price>
      </item>
      <item id="011">
         <description>Dishwasher</description>
         <price currency="swedish crown">10000.00</price>
      </item>
</inventory>
```

Parsing run

```
Start of element: inventory
Start of element: item
   Id attribute: 003
Start of element: description
Start of element: price
Start of element: item
   Id attribute: 007
Start of element: description
Start of element: price
Start of element: item
   Id attribute: 011
Start of element: description
Start of element: price
```

Here we handle only the “start element” event.
But we can watch out for other events...

```
module m_handlers
use flib_sax
private
public :: begin_element, end_element, pcdata_chunk
!
logical, private           :: in_item, in_description, in_price
character(len=40), private :: what, price, currency, id
!
contains !-----------------------------------------
!
subroutine begin_element(name,attributes)
..
subroutine end_element(name)
..
subroutine pcdata_chunk(chunk)
..
end module m_handlers
```

```
subroutine begin_element(name,attributes)
   character(len=*), intent(in)     :: name
   type(dictionary_t), intent(in)   :: attributes

   integer  :: status

   select case(name)
     case("item")
       in_item = .true.
       call get_value(attributes,"id",id,status)

     case("description")
       in_description = .true.

     case("price")
       in_price = .true.
       call get_value(attributes,"currency",currency,status)

   end select

end subroutine begin_element
```

```
subroutine pcdata_chunk(chunk)
   character(len=*), intent(in) :: chunk

   if (in_description) what = chunk
   if (in_price) price = chunk

end subroutine pcdata_chunk
```

The chunk of characters is assigned to a variable depending on context

We keep track of the context using logical variables

```
subroutine end_element(name)
   character(len=*), intent(in)     :: name

   select case(name)
     case("item")
       in_item = .false.
       write(unit=*,fmt="(5(a,1x))") trim(id), trim(what), ":", &
                                     trim(price), trim(currency)

     case("description")
       in_description = .false.

     case("price")
       in_price = .false.

   end select

end subroutine end_element
```

```
003 Washing machine : 1500.00 euro
007 Microwave oven : 300.00 euro
011 Dishwasher : 10000.00 swedish crown
```

- We handle parsing events.
- Context is monitored using logical module variables.
- Data is assigned to user variables depending on the context.
- Actions taken depending on context.
- Everything is done in one pass!

Parser features

- It checks well-formedness, reporting errors and giving the line and column where they occur.
- It understands the standard entities in PCDATA (&lt; &gt; , etc).
- Handles <![CDATA[&5<<!!(unparsed)]]> objects.
- It does not validate the document.

```
<?xml version="1.0"?>

<!DOCTYPE public test>
<!-- A Comment: Illustration of Allowable Constructs -->
<test version   =  "0.1">
<preamble>A small file exercising all the features
          in the parser...
</preamble>

<title>Mary had a &lt;little&gt; lamb who liked standard
entities</title>

<text>This is some text, with some cdata sections
inside to make it more interesting. How about this
           <![CDATA[
                   <begin>
                    pepe
                   </end> ]]>    ?
</text>
<single what="An 'empty' tag" />
</test>
```

Other (less useful) handlers

Processing instructions: `<?xml version="1.0"?>`

`xml_declaration_handler()`

Comments: `<!-- This is a comment -->`

`comment_handler()`

SGML declarations: `<!DOCTYPE public inventory ...>`

`sgml_declaration_handler()`

XML for scientific data

```
<data>
 8.90679398599 8.90729421510 8.90780189594 8.90831710494
 8.90883991832 8.90937041202 8.90990866166 8.91045474255
 8.91100872963 8.91157069732 8.91214071958 8.91271886986
 ...
 8.92100651514 8.92170571605 8.92241403816 8.92313153711
 8.92385826683 8.92459427943 8.92533962491 8.92609435120
 8.92685850416 8.92763212726 8.92841526149 8.92920794545
</data>
```

```
real, dimension(10000)  :: x    ! array to hold data
ndata = 0
...
subroutine pcdata_chunk_handler(chunk)
   character(len=*), intent(in) :: chunk

   if (in_data) call build_data_array(chunk,x,ndata)
   ...
end subroutine pcdata_chunk_handler
```

```
call build_data_array(pcdata_chunk,x,ndata)
```

- build_data_array fills the array with numbers of the correct kind, on the fly.
- Updates the counter for number of elements.
- Multidimensional arrays: can use reshape afterwards.

XPath interface

```
<inventory>
<item id="003">
  <description>Washing machine</description>
  <price currency="euro">1500.00</price>
</item>
</inventory>
```

```
PATH:  /inventory/item/description
```

```
//a         : Any occurrence of element 'a', at any depth.
/a/*/b      : Any 'b' which is a grand-child of 'a'
./a         : A relative path (to the current element)
a           : (same as above)
/a/b/./c    : Same as /a/b/c (the dot (.) is a dummy)
//*         : Any element.
//a/*//b    : Any 'b' under any children of 'a'.
```

```
program simple
use flib_xpath

type(xml_t) :: fxml
integer   :: status
character(len=100)   :: what

call open_xmlfile("inventory.xml",fxml,status)
!
do
  call get_node(fxml,path="//description", &
                     pcdata=what,status=status)
  if (status < 0)   exit
  print *, "Appliance: ", trim(what)
enddo
end program simple
```

get_node: select the next element node with given path

Collect the PCDATA in variable what

XPath standard is not fully supported

```
get_node(fxml,path,pcdata,attributes,status)
```

- /item Gets the next ‘item’ element, not all
- /item[3] Need to use get_node in a loop
- /item/price@currency (Need to look in attributes)
- /item/description/text() (Get and process pcdata)
- /item/price@currency=”euro” ....

Selection based on attribute values

```
program euros
use flib_xpath

type(xml_t) :: fxml

integer  :: status
character(len=100)  :: price

call open_xmlfile("inventory.xml",fxml,status)
!
do
  call get_node(fxml,path="//price", &
                att_name="currency",att_value="euro", &
                pcdata=price,status=status)
  if (status < 0)   exit
  print *, "Price (euro): ", trim(price)
enddo
end program euros
```

Contexts and relative searches

```
<inventory>
<item id="003">
  <description>Washing machine</description>
  <price currency="euro">1500.00</price>
</item>
</inventory>
```

```
!
call mark_node(fxml,path="//item",status=status)
if (status < 0)   exit        ! No more items
!
! Search relative to context
!
call get_node(fxml,path="price", &
    attributes=attributes,pcdata=price,status=status)
```

Contexts: Encapsulation of parsing tasks

```
call open_xmlfile("inventory.xml",fxml,status)
!
do
  call mark_node(fxml,path="//item",status=status)
  if (status /= 0)   exit      ! No more items
  call get_item_info(fxml,what,price,currency)
  write(unit=*,fmt="(5a))") "Appliance: ", trim(what), &
                            ". Price: ", trim(price), " ", &
                            trim(currency)
  call sync_xmlfile(fxml)
enddo
```

Routine to extract the price and description information from any element with the ‘item’ structure

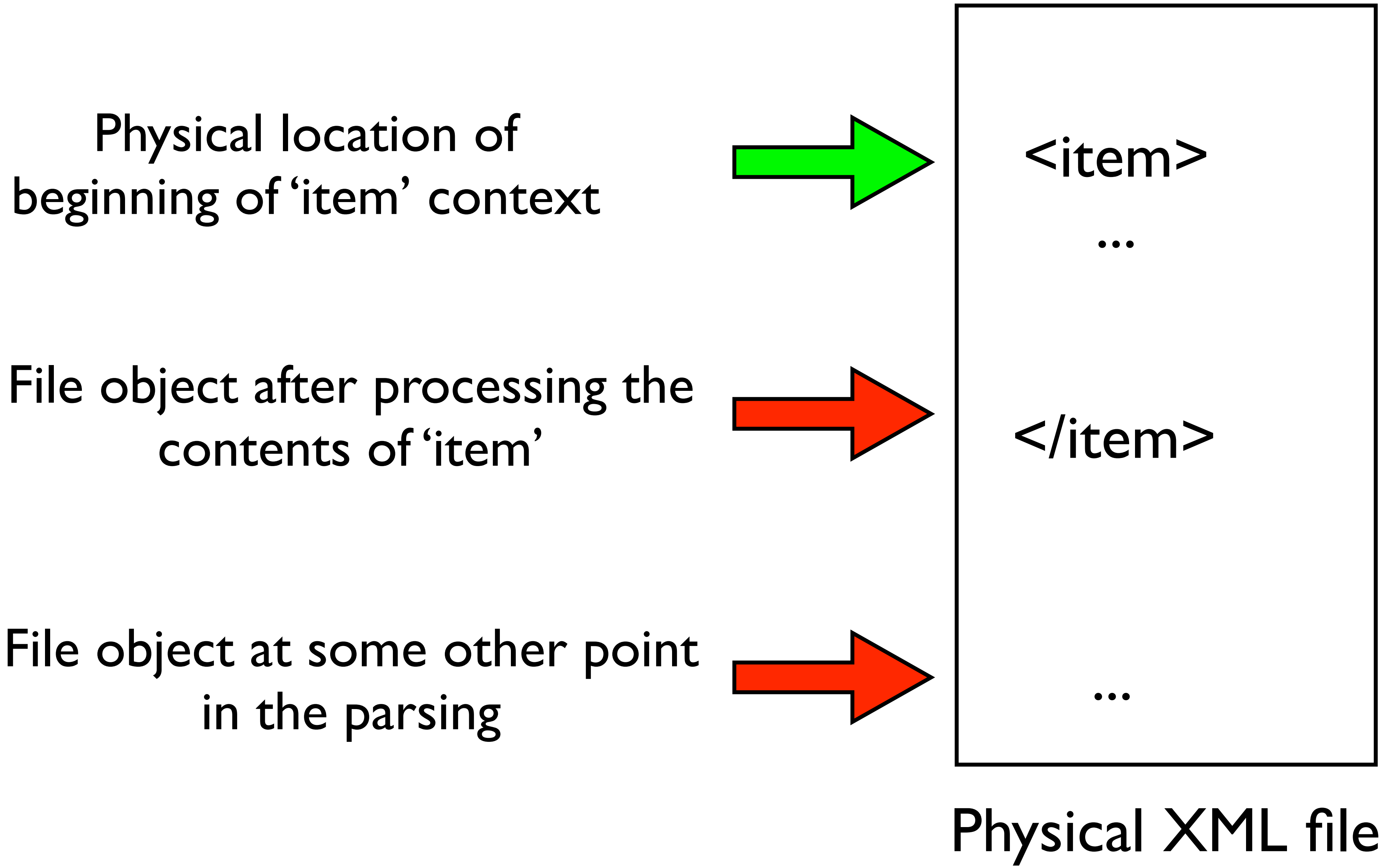
```
subroutine get_item_info(context,what,price,currency)
type(xml_t), intent(in)       :: context
character(len=*), intent(out) :: what, price, currency

  call get_node(context,path="price", &
                attributes=attributes,pcdata=price,status=status)
  call get_value(attributes,"currency",currency,status)
  !
  call get_node(context,path="description",
                pcdata=what,status=status)

end subroutine get_item_info
```

```
call sync_xmlfile(fxml)
```

- This version of XPATH is built on top of SAX: It inherits the stream paradigm.
- One has to be careful to go back to the appropriate place in the file to pick up the data within a given context.
- Alternative: digest all the information at once (use DOM as foundation) (Future?)



Physical location of beginning of 'item' context

File object after processing the contents of 'item'

File object at some other point in the parsing

```
<item>
...

</item>

...
```

Physical XML file

Use of Fortran derived types

```
<table units="nm" npts="100">
<description>Cluster diameters</description>
<data>
2.3 4.5 5.6 3.4 2.3 1.2 ...
...
...
</data>
</table>
```

```
type :: table
  character(len=50)            :: description
  character(len=20)            :: units
  integer                      :: npts
  real, dimension(:), pointer  :: data
end type table
```

Write a parsing method for each type !

Summary

- Process XML files directly in Fortran
- Choice of interface (SAX / XPath)
- Reasonably fast and memory-efficient
- Support for scientific data handling.
- Free and open to contributions.