Getting Started

J.J. Germishuizen

June 28, 2006

Contents

1. Introduction

Probably you have tried some of the source files immediatly after download. Got stuck and now look for help. This Getting Started will introduce the XML library written in Fortran [1]-[2], and is devided into three parts:

- compiling and testing the parser,
- using the *xmlreader* module to generate a reading subroutine and
- a simple addressbook example.

The Fortran parser allows programmers to access xml files without using mixed programming technices. An advantage of the parser is the *xmlreader* program which automatically generates a reading subroutine. This attractive feature allows any user to quickly get started using the parser. For each part a Fortran project needs to be created. The projects and its source files are given in App. A.

2. The Fortran XML parser

The programe readfile demonstrates a basic use of the parser module. Read any *.xml* and display all the tags found. How the program behaves when a certain tag is found is explained in section 3. The readfile.f90 and xmlparse.f90 are the driver and xml-parser library respectively. Variable *fname* in readfile.f90, of which the sourcecode is given in App. B, should be changed to the filename of the xml-file. The parser is tested reading the entries of an addressbook as given in section 2.1.

2.1. Testfile for parser

```
<?xml version="1.0"?>
<addressbook>
   <Entry Surname="Meier">
      <Name>Peter</Name>
      <Street>Waldweg</Street>
      <Code>10230</Code>
      <City>Waldesruh</City>
      <Tel>39840983</Tel>
      <email>peter.meier@xml.com</email>
   </Entry>
   <Entry Surname="Cornelissen">
      <Name>Johanette</Name>
      <Street>Suikerbossie</Street>
      <Code>7646</Code>
      <City>Brakpan</City>
      <Tel>22879631</Tel>
      <email>johanette.cornelissen@xml.com</email>
   </Entry>
</addressbook>
```

3. The xmlreader module

Once you have designed the structure of the XML file, a reading subroutine is necessary to treat the input data accordingly. This can be time consuming. The xmlreader module is a help to automatically generate such a reading subroutine. A template[1] of the actual data has to be created by the user that will be the input for the xmlreader program. As the template is a xml file it can be viewed using any xml-viewer.

For our addressbook example as given in section 2.1 we need to create a template, called addressbook_t.xml[2]. A screenshot of the template, using Mozilla Firefox, is shown

[1] The template itself is also a xml-file of which the rootelement is defined as template.

[2] The actual file containing all the addresses is called addressbook.xml. For convinience and for later refenrence we just add _t to the actual filename. This then indicates that it is the template for addressbook.xml. The template file could however have any other name.

```
<?xml version="1.0" ?>
- <template>
    <options strict="yes" dynamicstrings="no" rootname="addressbook" />
  - <typedef name="entry_t">
      <component name="Surname" type="word" length="40" />
      <component name="Name" type="word" length="40" />
      <component name="Street" type="line" length="40" />
      <component name="Code" type="integer" />
      <component name="City" type="line" length="40" />
      <component name="Tel" type="integer" />
      <component name="email" type="line" length="40" />
    </typedef>
    <variable name="Entry" type="entry_t" dimension="1" />
  </template>
```

Figure 1: xmlreader template: addressbook_t.xml

in Fig. 1. Some options can be set using the options specification. For the example we would like the parser to test for unknown tags. This is done using strict="yes". Because some Fortran compilers do not support a feature of Fortran 95 that is used by default, a dynamic lenght for local character variables in a subroutine, we turn this feature off using dynamicstrings="no". Instead of dynamic strings we allow for each string a length of 40. Further, we set the rootname of the addressbook by rootname="addressbook".

The xmlreader program has an inputfile, xmlreader.inp. This file contain the filename of the template. Only the filename, without the extension, is necessary. The xmlreader program will read this template and generate a reading module having a name of the form xml_data_addressbook_t in addressbook_t.f90. This file must be included in the final project.

3.1. Reading addresbook entries

The reading subroutine generated using the template in Fig. 1 can now be used to read the addressbook in section 2.1. For reading the addressbook entries in addressbook.xml the main program is given in App. C.

4. Summary

In this Getting started the minimum to get the Fortran xml parser running was explained. An easy to understand addressbook example was used to show the use of the xml reader program for automatically generating a module for treating the data.

A. Project files

The project files for each of the projects explained are given in Table 1.

Project	Parser	xmlreader	Addressbook
1	readfile.f90	xmlread.f90	tst_addressbook.f90
2	xmlparse.f90	xmlparse.90	addressbook_t.f90
3		xmlreader.inp	xmlparse.f90
4			read_xml_prims.f90
5			read_from_buffer.inc
6			read_xml_scalar.inc
7			ream_xml_array.inc

Table 1: Project files

B. Testprogram sourcecode - readfile.f90

```
program readfile
  use xmlparse
  character(len=20) :: fname
  logical           :: mustread
  type(XML_PARSE)   :: info
  character(len=80)                       :: tag
  logical                                 :: endtag
  character(len=80),dimension(1:2,1:20) :: attribs
  integer                                 :: no_attribs
  character(len=200),dimension(1:100)   :: data
  integer                                 :: no_data
  integer                                 :: i
! Assign the xml-filename to fname and open the file
  mustread = .true.
  fname    = 'adressbook.xml'
  call xml_open(info,fname,mustread)
! Check for errors
  if (xml_error(info)) then
!   handle the errors
  else
!   Start reading the file
    call xml_options(info,ignore_whitespace = .true.)
    do
      call xml_get(info,tag,endtag,attribs,no_attribs,data,no_data)
      if (xml_error(info)) then
!       handle the errors
```

```
      endif
      write(*,*) tag,endtag
      do i=1,no_attribs
        write(*,*) i,'>',attribs(1,i),'<=',trim(attribs(2,i))
      enddo
      write(*,*) (i,'>',trim(data(i)),'<',i=1,no_data)
      if (.not. xml_ok(info)) exit
    enddo
  endif
  call xml_close(info)
  stop
end program
```

C. Main program - tst_addressbook.f90

```
! Test program for generated code
program addressbook
   use xml_data_addressbook
   integer :: n,i
   call read_xml_file_addressbook( 'addressbook.xml' )
   n = size(Entry)
   do i=1,n
     write(*,'(A40)')    Entry(i)%Surname
     write(*,'(A40)')    Entry(i)%Name
     write(*,'(A40)')    Entry(i)%Street
     write(*,'(I10.10)') Entry(i)%Code
     write(*,'(A40)')    Entry(i)%City
     write(*,'(I10.10)') Entry(i)%Tel
     write(*,'(A40/)')   Entry(i)%email
   enddo
end program
```

References

[1] Markus, A.: Have a look at XML files, ACM SIGPLAN Fortran Forum, Volume 23 Nr. 3, pp. 2-10, December 2004

[2] Markus, A. : Have a second look at XML files, ACM SIGPLAN Fortran Forum, Volume 24 Nr.1, pp. 2-5, April 2005

[3] xmlparse.html: Online documentation, Generating a reading subroutine