IOTK binary files

Giovanni Bussi

February 17, 2016

The input/output tool kit (IOTK) is a Fortran 90 library designed to help input/output formatting. The library is able to handle textual files and binary files with an identical interface. The syntax of textual files is based on XML. In this short document I explain how to map the syntax of the binary files onto the textual format. This translational rules are used inside the IOTK library, but could be easily implemented in other parsers.

Of course these binary files are not portable across platforms. However, since there is a one-to-one correspondence between this format and textual XML files, they can be converted to text and compressed for porting or archiving, while they can be used directly when the I/O performance is an issue, e.g. for restart files.

1 Fortran binary files

A IOTK binary file is a sequential-access, binary Fortran file. It is basically a sequence of binary records, each of them accessed through a single Fortran I/O statement. When reading a Fortran file, one is able to skip a record using an empty `read` command, or even to read just the beginning of a record of unknown length. That is, if one writes a file with

```
write(10) a,b
write(10) c,d
```

and is interested in retrieving the value of `a` and `c` only, they can be read from the file with

```
read(10) a
read(10) c
```

To obtain this flexibility, the Fortran I/O library writes two record markers, one before the record and one after it. This record markers are usually 4 bytes integers indicating the length of the record, expressed in bytes. In the following, I will describe the file format in term of Fortran statements, and I will never include in the description these implicit record markers. The only effect of these markers is the Fortran ability to skip part of the data in a record.

2 Tags in IOTK binary files

Tags are represented in IOTK binary files directly as character strings. That is, the tag `<pippo a="1">` will be written as a string `'<pippo a="1">'`. The only additional information which is required to be stored in the file is the length of the tag itself, so that it can be read back. The IOTK library also stores informations about the type of the written tag (begin tag, end tag, comment tag, etc.). These informations are useful to know the type of a tag without the need of parsing the tag string.

Each tag is written to a file as a pair of records.[1] The two records representing a tag are written as

```
integer(iotk_header_kind) :: header1,header2
write(unit) header1
write(unit) header2,trim(tag)
```

The kind of the headers is chosen to have 4 bytes integers.[2] The information about the type of tag is represented as a control code, defined using the following conventions:

[1]The decision of writing two successive records is motivated by the observation that (a) Fortran needs to know in advance the length of a record to read it entirely and (b) backspace operations are to be avoid to not affect the performance of the I/O action.

[2]It can be configured in the IOTK configuration file, the default being `selected_int_kind(8)`

control code	meaning	example tag
1	begin	`<pippo>`
2	end	`</pippo>`
3	empty	`<pippo/>`
4	comment	`<!--pippo-->`
5	processing instruction	`<?pippo?>`
128	(used for `header2`)	—

The integer values of `header1` and `header2` incorporate both the control code and the tag length as

```
header1 = control + 256*len_trim(tag)
header2 = 128     + 256*len_trim(tag)
```

so that the tag length and the control code can be extracted from the first header as

```
control = modulo(header1,256)
taglen  = header1/256
```

and can be used to properly read the second record. The second header can be used for a check.

Note that the `taglen` variable represents the length of the string written on the file, so that the length of the second record will be `taglen` *plus* the four bytes for the header. This string can be longer than the real tag, since the library allows additional arbitrary characters before and after the tag. Any character before the `<` symbol and after the `>` symbol is removed on reading. This allows to put extra characters to help formatting before and after the tag itself. The library presently adds a newline character plus a variable number of space before the tag and an additional newline character after the tag. That is, the tag `<pippo a="1">` will be written as

```
tag='<pippo a="1">'
tagwrite='\n  '//trim(tag)//'\n'
! the number of spaces before the tag is arbitrary
write(unit) 1 + 256*len_trim(tagwrite)
! 1 is the control code for a begin tag
write(unit) 128 + 256*len_trim(tagwrite),trim(tagwrite)
```

The newlines help if one wants to use line-based tools like grep and sed to scan the binary files.[3] The extra spaces can be used for tag indentation. Reading that tag is easily done with

```
read(unit) header1
control = modulo(header1,256)
! the control code carries the information concerning the type of the tag
taglen  = header1/256
read(unit) header2,tagread(1:taglen)
select case(control)
case(1)
  predelim="<" ; postdelim=">"
case(2)
  predelim="</" ; postdelim=">"
case(3)
  predelim="<" ; postdelim="/>"
case(4)
  predelim="<!--" ; postdelim="-->"
case(5)
  predelim="<?" ; postdelim="?>"
end select
tag=tagread((index(tagread,trim(predelim))): &
            (index(tagread,trim(postdelim))))
```

The tag now is ready to be parsed with a standard XML parser.

3 Data in IOTK files

The representation of data in IOTK files is self-describing. That is, any intrinsic Fortran object written in the file should carry informations about its type and kind, plus size for arrays and length for strings. Using these informations, an external tool which does not know anything about the content should be able to reformat the data, that is to bring them from the binary representation to the XML textual one. Moreover, when the IOTK library is used to read the data back, it is able to check size consistency and to perform transparent kind conversion.

[3]Inside the IOTK library, the newline is represented as `achar(10)`. However, since it is skipped on reading, any character other then `<` or `>` could be used.

In this context an intrinsic Fortran object is a scalar or an array of an intrinsic type and of any possible kind. In the textual representation it is written as the value surrounded by a pair of tags:

```
<pippo type="real" size="3">
  1.0 2.0 3.0
</pippo>
```

Note that in this textual representation there is no need to include kind informations. In the binary format, the tags are written explicitly as strings, as explained in the previous section, while the data are written in binary format, gaining a factor of approximately 3 in the size of the file and also with a strong speed improvement (there is no need of conversions). The same object as before is written as

```
! begin tag:
begin_tag='\n  <pippo type="real" size="3" kind="4">\n'
write(unit) 1+256*len_trim(begin_tag)
write(unit) 128+256*len_trim(begin_tag),begin_tag

! data itself, note the 0 preceding the data
write(unit) 0 , (/1.0,2.0,3.0/)

! end tag:
end_tag='\n  </pippo>\n'
write(unit) 2+256*len_trim(end_tag)
! 2 is the control code for end tags
write(unit) 128+256*len_trim(end_tag),end_tag
```

Note the 0 written in front of the data itself. This is needed when the library scans the file to distinguish between tags (which have a header always different from 0) and data.

When dealing with strings, the iotk library stores also its length[4], as in the following

```
<string type="character" size="3" len="7">
uno
due
```

[4]The number of characters in a string array is length times size.

```
tre
quattro
</string>
```

The Fortran instructions to write its binary representation are

```
! begin tag:
begin_tag='\n  <string type="character" size="3" len="7">\n'
write(unit) 1+256*len_trim(begin_tag)
write(unit) 128+256*len_trim(begin_tag),begin_tag

! data itself, note the 0 preceding the data
write(unit) 0 , (/"uno    ","due    ", &
                  "tre    ","quattro"/)
! end tag:
end_tag='\n  </string>\n'
write(unit) 2+256*len_trim(end_tag)
write(unit) 128+256*len_trim(end_tag),end_tag
```

To read these objects one has first to read the begin tag, then to parse it to obtain the informations about the type and size (for checking), kind (for eventual transparent conversion). For characters, also the len is read. The parsing needs to be done with some XML parser, then the reading of the data record can be done directly from Fortran, just keeping in mind to skip the dummy integer. In the IOTK library, all these operations are hidden inside a single library call.