DCMTK - Bug #807

Some methods expect getVM() to always return the number of stored values (which is not correct)

2017-12-13 17:31 - Jörg Riesmeier

Status:	Closed	**Start date:**	2017-12-14
Priority:	High	**Due date:**	
Assignee:		**% Done:**	100%
Category:	Library	**Estimated time:**	0:00 hour
Target version:	3.6.4		
Module:	dcmdata	**Compiler:**	
Operating System:			

Description

As documented in the various VR classes, getVM() always returns the Value Multiplicity, which is 1 by definition for some VRs (e.g. OF, OD and OL). However, methods like compare() and matches() expect getVM() to return the number of stored values, which is not always true, and, therefore, create wrong results.

Related issues:

Follows DCMTK - Feature #808: Introduce a new method getNumberOfValues() or g...	**Closed**	**2017-12-13**

History

#1 - 2017-12-13 17:36 - Jörg Riesmeier

- Follows Feature #808: Introduce a new method getNumberOfValues() or getValueCount() for the various VR classes added

- Due date set to 2017-12-14

- Start date changed from 2017-12-13 to 2017-12-14

#2 - 2018-01-17 11:18 - Marco Eichelberg

- Priority changed from Normal to High

#3 - 2018-02-05 18:40 - Jan Schlamelcher

- Target version changed from 3.6.3 to 3.6.6

#4 - 2018-02-27 18:18 - Jörg Riesmeier

- Assignee set to Jörg Riesmeier

- Target version changed from 3.6.6 to 3.6.4

#5 - 2018-03-02 12:42 - Michael Onken

The situation is a different and kind of inverted to what is described above.

Most (all?) VR classes that are in the toolkit (and DICOM) for a long time define the getVM() method in a way, that returns the number of values in the value field. This includes AT, SS, SL, US, but also Bytestrings (AE, AS, DA, etc.) and others.

However, since "always" there are also exceptions for DcmItem and DcmSequence (and DcmPixelSequence, maybe other derived classes), that always return "1" on getVM() (i.e. as defined in the standard, or if you like, the DICOM dictionary). Those classes offer a card() method to get the number of values (elements, items) inside.

Some newer VR classes added in recent years in DICOM and DCMTK (those mentioned above: OF,OD,OL, others?) derive from existing classes that use the "count values" approach, however, they re-define getVM() to return 1 as denoted in the DICOM dictionary/standard for those VRs.

The question is how to go from here. The most important point should be to have a consistent implementation aftewards, and not to break most existing user implementations.

Due to the long term usage of getVM() in the sense of "count values", existing implementations expect exactly that from the method. One could change the behaviour on DcmSequenceOfItems's and DcmItem's getVM() implementation to be consistent. Probably that does not affect many users since they presumably do not use the method (it always returns 1), instead, most users will already use card(), or if they iterate over DcmElements or DcmObjects and need a general approach, they already will do a dictionary lookup. One also could keep these both exceptions since they always have been there. Changing the behaviour of getVM() to generally return the number of values, however, will break many existing implementations, since this a a funcationality that is probably used a lot, also when looping over DcmObjects/Elements. DcmItem and DcmSequenceOfItems are special anyway, so they are usually handled separately in user (and DCMTK code, e.g. by using isLeaf() to differentiate).

What I dont like is that the derived methods of OF, OD and OL re-implement VM from their base classes with a deviating behaviour, and since they are quite new in DCMTK (compared to all other VR classes), they should stick with the existing implementation behaviour. At the same time one could add a method called "getDictVM()" or something similar that does

So my proposoal is the following:

- getVM() should return the number of values in the value field (i.e. fix OF, OD, and OL)
- DcmSequenceOfItems and DcmItem should follow this rule, though this can be discussed.
- A separate method (getDictVM()), if needed at all (people probably already use dictionary lookup), could return the VM as defined by the standard/dictionary. This could happen on level DcmTag which already does a dictionary lookup to serve getTagName() and VR lookup.

#6 - 2018-03-02 13:44 - Jörg Riesmeier

The assumption that getVM() ever returned anything else than Value Multiplicity (of the data element) according to the DICOM standard is not correct. Also for VRs that are part of the DICOM standard since 1993 (e.g. OB and OW) the return value of getVM() is and always was 1 (see here). Same for ST, LT, UT, ...

Furthermore, getVM() does not and never did return the VM of the data dictionary (another wrong assumption), so a new method getDictVM() also makes no sense.

#7 - 2018-03-02 14:08 - Michael Onken

```
The assumption that getVM() ever returned anything else than Value Multiplicity (of the data element) accordin
g to the DICOM standard is not correct.
```

Many or all of the VRs I listed implement getVM() by counting values and not returning the static VM information found in the standard.

```
Furthermore, getVM() does not and never did return the VM of the data dictionary (another wrong assumption)
```

I guess you know what was meant: Returning static VM information from the standard. Sometimes the information is also in the dictionary, though it is not determined by the dictionary but originally by the VR rules in the standard.

```
so a new method getDictVM() also makes no sense.
```

Choose any name you like that fits better.

So what do you recommend? Actually changing all the "counting" getVM() to...what? If the method does something different in the future, I'm sure this will break many user implementations. If you remove it, it is not very consistent in the object hierarchy. The "hybrid" approach of the method sometimes counting values and sometimes providing static information from the standard should not be the future solution in my opinion.

#8 - 2018-03-02 14:12 - Jörg Riesmeier

My proposal is referenced as Feature #808: introduce a new method getValueCount() and leave getVM() as is (and as it always was: return the Value Multiplicity of the data element not of the attribute according to DICOM part 6; the latter should be handled by the data dictionary class).

#9 - 2018-03-02 14:13 - Jörg Riesmeier

By the way, getVM() is used in dcmdump's output (i.e. what the print() method does). This has nothing to do with the data dictionary but with the definitions according to DICOM part 5!

#10 - 2018-03-02 14:39 - Michael Onken

Ok, after discussing this in a call I got the intention behind Jörg's argumentation: getVM() always returns what DICOM defines to be the VM of a value field of a certain VR. For some VRs this is fixed to 1, for others this requires counting values. So the method as implemented on the VRs right now does the right thing by sometimes counting, sometimes using the pre-defined constant in the standard.

Since it is probably confusing for users who look for a "count values" method on the DcmObject (root) class interface and end up by getVM() which only partly does what they (spuriously) expect, we should enhance the documentation on getVM(), at least on DcmObject, where users probably start to look for such a method.

#11 - 2018-03-02 17:03 - Jörg Riesmeier

- Due date deleted (2017-12-14)

- % Done changed from 0 to 10

Added sentence to API documentation of DcmObject::getVM() explaining that depending on the VR some subclasses return the currently stored number of values and others the constant value 1 (according the DICOM standard). See commit ac53541.

#12 - 2018-03-12 15:18 - Jörg Riesmeier

- Assignee deleted (Jörg Riesmeier)

- % Done changed from 10 to 50

Partly closed by commit adc2131.

#13 - 2018-03-21 12:09 - Michael Onken

- % Done changed from 50 to 90

Fixed in dcmdata (compare() methods) in commit a61651.
Fixed getVM() usage modules dcmiod, dcmfg and dcmtract in commit e383e4.

Correct usage of getVM() must still be checked in methods matches() and isUniversalMatch() (dcmdata module).

#14 - 2018-11-15 19:39 - Jörg Riesmeier

Methods matches() and isUniversalMatch() should be checked for partly useless code like the following:

```
OFBool DcmUniversalResourceIdentifierOrLocator::isUniversalMatch(const OFBool normalize,
                                                                 const OFBool enableWildCardMatching)
{
  if(!isEmpty(normalize))
  {
    if(enableWildCardMatching)
    {
      OFString value;
      if(!normalize && getVM() > 1)
        return OFFalse;
      for(unsigned long valNo = 0; valNo < getVM(); ++valNo)
      {
        getOFString(value, valNo, normalize);
        if(value.find_first_not_of( '*' ) != OFString_npos)
          return OFFalse;
      }
    }
    else
      return OFFalse;
  }
  return OFTrue;
}
```

Useless because of this:

```
unsigned long DcmUniversalResourceIdentifierOrLocator::getVM()
{
    /* value multiplicity is 1 for non-empty string, 0 otherwise */
    return (getRealLength() > 0) ? 1 : 0;
}
```

#15 - 2018-11-20 19:18 - Jörg Riesmeier

- Status changed from New to Closed

- % Done changed from 90 to 100

Finally closed by commit 6154c4e.