Subject: Re: read XML with IDL
Posted by Karl[1] on Thu, 15 Jul 2010 16:57:43 GMT
View Forum Message <> Reply to Message

On Jul 15, 8:05 am, Paulo Penteado <pp.pente...@gmail.com> wrote:
> On Jul 15, 5:41 am, sh <sebastian.h...@gmail.com> wrote:
>
>> But you have to overwrite some methods of the SAX parser to make use
>> of it for a certain xml file.
>> And I didn't want to write everytime a new xml_to_struct__define.pro!
>
>> Therefore I asked if there is a "general" solution to take the
>> elements/node/attributes names within a xml file to generate a
>> structur, like in my example above. Because in this case I have only
>> one parser who generates the output structure with a dependency on the
>> xml elements/node/attributes names and I can read every xml file!
>
> With IDL 8, it does not seem too difficult to write such a general
> parser to generate a hash. Or, for something more complicated to
> write, a class that inherited from the DOM class and presented a hash-
> like interface through overloaded [].
>
> Building a structure as the tree is parsed would always be awkward,
> and the result would be difficult to edit if it involved removing
> fields. But still there are situations where a structure would be more
> desirable as the end product, so that is one example of the need for a
> routine to create a structure from a hash, but I do not know if one
> will be provided.

One way to deal with an arbitrary XML file in IDL without doing
lexical-level parsing is to use IDLffXMLDOM. You can walk the entire
DOM tree, collecting elements and attributes, while maintaining
whatever relational information between them you need. When finished,
you can then generate the IDL structures to match the data
organization in the XML file. The important thing is that you don't
need to know the XML structure ahead of time in order to do this. I
think that it is possible to write a general routine for this that
would work for all XML input.

Just part of the algorithm would be:

Use XMLDOM methods to find all the element nodes. From this list find
all element nodes that do not have element nodes in them. These are
the "leaf" nodes. Then use IDL functions to make IDL structures for
each leaf element. The name of the struct is the name of the
element. Add element members for each attribute and the value of the
element.

For the non-leaf elements, you'll have to go back and build IDL
structs for these that contain structs for the leaf elements.

There are more details of course, but this looks like it could be
implemented fairly elegantly using recursion.

Anyway, all this strikes me as a bit odd. Most programs processing
XML input have a pretty good idea of the structure of the XML data
they are supposed to be reading. XML Schemas take this to the
extreme, where schema enforcement can be enabled to reject XML files
that do not conform to the schema. Finally, applications that read
XML data usually have some plan as to what they will do with the data
and won't know what to do with elements that they are not coded to
handle. But I guess that is the app's problem.

Also, processing an XML file in order to "learn" its schema is
problematic. You may not be able to discern the entire schema just by
reading one sample file. Taking the "planets" example, one might
suppose that a planet element may contain zero or more moon elements:

```
<planet name=Mercury>
</planet>
<planet name=Earth>
<moon name=Moon>
</moon>
</planet>
```

You'd have to be careful here when defining the planet IDL structure.
If you define it based on the first planet you encounter, you're going
to have to redefine it later (big problem) or make your "schema
discovery" code much more robust.

I'm just trying to say that it isn't typical (I think) to write XML
parsing code without integrating in some solid schema for the data.
If you want to do it anyway, the XMLDOM class allows you the maximum
flexibility in "discovering" your data on the fly.