
Subject: XML Parser has no support for DTD
Posted by [cbpporter](#) on Thu, 03 Jan 2008 20:32:37 GMT
[View Forum Message](#) <> [Reply to Message](#)

Hi!

I noticed that the XML parser has absolutely no support for DTD.

Example:

```
<?xml version="1.0"?>
<!DOCTYPE note [
  <!ELEMENT note (to,from,heading,body)>
  <!ELEMENT to (#PCDATA)>
  <!ELEMENT from (#PCDATA)>
  <!ELEMENT heading (#PCDATA)>
  <!ELEMENT body (#PCDATA)>
]>
<note>
  <to>Tove</to>
  <from>Jani</from>
  <heading>Reminder</heading>
  <body>Don't forget me this weekend</body>
</note>
```

I need this to be recognized, so I'm going to add this functionality. If you are going to add it ASAP, then I'll wait. If not, maybe I can contribute it.

Subject: Re: XML Parser has no support for DTD
Posted by [mirek](#) on Thu, 03 Jan 2008 20:36:24 GMT
[View Forum Message](#) <> [Reply to Message](#)

What exactly do you expect from DTD support?

Mirek

Subject: Re: XML Parser has no support for DTD
Posted by [cbpporter](#) on Thu, 03 Jan 2008 20:39:05 GMT
[View Forum Message](#) <> [Reply to Message](#)

Well, in that example:

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

```
<!DOCTYPE note [  
  <!ELEMENT note (to,from,heading,body)>  
  <!ELEMENT to    (#PCDATA)>  
  <!ELEMENT from  (#PCDATA)>  
  <!ELEMENT heading (#PCDATA)>  
  <!ELEMENT body  (#PCDATA)>  
>  
is interpreted as the following tags:  
<!DOCTYPE note [<!ELEMENT note (to,from,heading,body)>  
<!ELEMENT to    (#PCDATA)>  
<!ELEMENT from  (#PCDATA)>  
<!ELEMENT heading (#PCDATA)>  
<!ELEMENT body  (#PCDATA)>  
and the text  
>
```

Subject: Re: XML Parser has no support for DTD
Posted by [mirek](#) on Thu, 03 Jan 2008 20:42:14 GMT
[View Forum Message](#) <> [Reply to Message](#)

Sorry, I was not quite specific I think.

Basically, there are two options:

- 1) Skip it (which IMO should be quite simple)
- 2) Interpret it and ensure that XML follows

I agree than 1) is a must. I would be happy if you contribute that.

Mirek

Subject: Re: XML Parser has no support for DTD
Posted by [cbpporter](#) on Thu, 03 Jan 2008 20:44:37 GMT
[View Forum Message](#) <> [Reply to Message](#)

Quote:I agree than 1) is a must. I would be happy if you contribute that.
That's all what I need for now and I'll be happy to contribute. Do you want somekind of flag to turn on/off the DTD recognition?

Subject: Re: XML Parser has no support for DTD
Posted by [mirek](#) on Thu, 03 Jan 2008 20:46:43 GMT

[View Forum Message](#) <> [Reply to Message](#)

cbpporter wrote on Thu, 03 January 2008 15:44Quote:I agree than 1) is a must. I would be happy if you contribute that.

That's all what I need for now and I'll be happy to contribute. Do you want somekind of flag to turn on/off the DTD recognition?

IMO, no need. Just skip the damned thing

Mirek

Subject: Re: XML Parser has no support for DTD
Posted by [cbpporter](#) on Fri, 04 Jan 2008 08:53:32 GMT

[View Forum Message](#) <> [Reply to Message](#)

luzr wrote on Thu, 03 January 2008 21:46

IMO, no need. Just skip the damned thing

Mirek

The fix was easy. I did ignore it, just inserted the whole thing into a "nested" tag content, so in the previous example you would get a node with type of XML_DECL and content:

```
DOCTYPE note [  
  <!ELEMENT note (to,from,heading,body)>  
  <!ELEMENT to    (#PCDATA)>  
  <!ELEMENT from  (#PCDATA)>  
  <!ELEMENT heading (#PCDATA)>  
  <!ELEMENT body  (#PCDATA)>  
]
```

I tested it with the provided XmlViewer on this example and some other ones, including a 12MB XML, and it seems to work fine with well formatted XMLs.

Which brings me to my next problem. The XML parser code looks pretty efficient, and while it could be improved, the results wouldn't be that different. And loading that 12MB XML takes a lot (but less than in other programs I tried capable of loading an XML tree). I need a way to speed up thing, and the only thing I can think right now is a binary cache of the XML data.

Here is my fix:

```
void XmlParser::Next()  
{  
  nattr.Clear();  
  nattr1 = nattrval1 = Null;  
  if(empty_tag) {  
    empty_tag = false;  
    type = XML_END;  
    return;  
  }
```

```

}
text.Clear();
type = XML_EOF;
if(!npreserve)
    SkipWhites();
if(*term == '<') {
    term++;
    if(*term == '!') {
        type = XML_DECL;
        term++;
        if(term[0] == '-' && term[1] == '-') {
            type = XML_COMMENT;
            term += 2;
            for(;;) {
                if(term[0] == '-' && term[1] == '-' && term[2] == '>')
                    break;
                if(*term == '\0') {
                    throw XmlError("Unterminated comment");
                    type = XML_EOF;
                    return;
                }
                if(*term == '\n')
                    line++;
                text.Cat(*term++);
            }
            term += 3;
            return;
        }
        bool intdt = false;
        for(;;) {
            if (*term == '[')
                intdt = true;
            if(*term == '>' && intdt == false) {
                term++;
                break;
            }
        }
        if(intdt == true && term[0] == ']' && term[1] == '>') {
            text.Cat(*term++);
            term++;
            break;
        }
        if(*term == '\0')
            throw XmlError("Unterminated declaration");
        if(*term == '\n')
            line++;
        text.Cat(*term++);
    }
}
}

```

```

else
if(*term == '?') {
    type = XML_PI;
    term++;
    for(;;) {
        if(term[0] == '?' && term[1] == '>') {
            term += 2;
            return;
        }
        if(*term == '\0')
            throw XmlError("Unterminated processing info");
        if(*term == '\n')
            line++;
        text.Cat(*term++);
    }
}
else
if(*term == '/') {
    type = XML_END;
    term++;
    const char *t = term;
    while(IsXmlNameChar(*term))
        term++;
    text = String(t, term);
    if(*term != '>')
        throw XmlError("Unterminated end-tag");
    term++;
}
else {
    type = XML_TAG;
    const char *t = term;
    while(IsXmlNameChar(*term))
        term++;
    text = String(t, term);
    for(;;) {
        SkipWhites();
        if(*term == '>') {
            term++;
            break;
        }
        if(term[0] == '/' && term[1] == '>') {
            empty_tag = true;
            term += 2;
            break;
        }
        if(*term == '\0')
            throw XmlError("Unterminated tag");
        const char *t = term++;
    }
}

```

```

while((byte)*term > ' ' && *term != '=' && *term != '>')
    term++;
String attr(t, term);
SkipWhites();
if(*term == '=') {
    term++;
    SkipWhites();
    StringBuffer attrval;
    if(*term == "\\") {
        term++;
        while(*term && *term != "\\")
            if(*term == '&')
                Ent(attrval);
            else {
                const char *e = term;
                while(*++e && *e != '&' && *e != "\\")
                    ;
                attrval.Cat(term, (int)(e - term));
                term = e;
            }
        if(*term == "\\")
            term++;
    }
    else {
        while((byte)*term > ' ' && *term != '>')
            if(*term == '&')
                Ent(attrval);
            else {
                const char *e = term;
                while((byte)*++e > ' ' && *e != '>' && *e != '&')
                    ;
                attrval.Cat(term, (int) (e - term));
                term = e;
            }
    }
    if(attr == "xml:space" && attrval.GetLength() == 8 && !memcmp(~attrval, "preserve", 8))
        npreserve = true;
    String aval = FromUtf8(~attrval, attrval.GetLength()).ToString();
    if(IsNull(nattr1)) {
        nattr1 = attr;
        nattrval1 = aval;
    }
    else
        nattr.Add(attr, aval);
}
}
}
}
}

```

```

else if(*term == 0)
    type = XML_EOF;
else {
    StringBuffer raw_text;
    while(*term != '<' && *term) {
        if(*term == '\n')
            line++;
        if(*term == '&')
            Ent(raw_text);
        else {
            const char *e = term;
            while(*++e && *e != '&' && *e != '<')
                ;
            raw_text.Cat(term, (int)(e - term));
            term = e;
        }
    }
    const char *re = raw_text.End();
    if(!Inpreserve) {
        while(re > raw_text.Begin() && (byte)re[-1] <= ' ')
            re--;
    }
    text = FromUtf8(~raw_text, (int)(re - ~raw_text)).ToString();
    type = XML_TEXT;
}
}

```