
Subject: CParser: proposal of new functions
Posted by [omari](#) on Wed, 16 May 2012 16:40:32 GMT
[View Forum Message](#) <> [Reply to Message](#)

Hello ,

I have to load data in a Vector<TestCase>
the struct of the data file is:

TEST CASE:

Name of Test case: TC_0000
Test case version: 1
Author:
Date: 10/05/2012
Test type: nominal
SRD_REQUIREMENT: SRD-REQ-0628
SRD version: G02
Precondition:
...
Description:
visit LoginPage
enter userID
....
Expected Result:
...

END OF TEST CASE

to do that i use CParser and i have added a couple of function:

in CParser.h:

```
bool IsId2(const char *s1, const char *s2) const;  
bool IsId3(const char *s1, const char *s2, const char *s3) const;  
bool Id2(const char *s1, const char *s2);  
bool Id3(const char *s1, const char *s2, const char *s3);  
String ReadId2() throw(Error);  
String ReadId3() throw(Error);  
String ReadUntil(int delim);  
String ReadLine() {return ReadUntil('\n');}
```

the function ReadUntil is based on ReadOneString()

if you find this functions useful, i would be glad if you add them to CParser..

in CParser.cpp:

```

bool CParser::IsId2(const char *s1, const char *s2) const
{
    CParser p = *this;

    if (!p.Id(s1)) return false;

    return p.IsId(s2);
}

```

```

bool CParser::IsId3(const char *s1, const char *s2, const char *s3) const
{
    CParser p = *this;

    if (!p.Id(s1)) return false;
    if (!p.Id(s2)) return false;

    return p.IsId(s3);
}

```

```

bool CParser::Id2(const char *s1, const char *s2)
{
    CParser p = *this;

    if (!p.Id(s1)) return false;
    if (!p.Id(s2)) return false;

    *this = p;

    return true;
}

```

```

bool CParser::Id3(const char *s1, const char *s2, const char *s3)
{
    CParser p = *this;

    if (!p.Id(s1)) return false;
    if (!p.Id(s2)) return false;
    if (!p.Id(s3)) return false;

    *this = p;

    return true;
}

```

```
String CParser::ReadId2() throw(Error)
{
    String ret = "";

    ret << ReadId();
    const char* p = GetSpacePtr();
    ret << String(p, term - p);
    ret << ReadId();

    return ret;
}
```

```
String CParser::ReadId3() throw(Error)
{
    String ret = "";

    ret << ReadId();
    const char* p = GetSpacePtr();
    ret << String(p, term - p);
    ret << ReadId();
    p = GetSpacePtr();
    ret << String(p, term - p);
    ret << ReadId();

    return ret;
}
```

```
String CParser::ReadUntil(int delim)
{
    StringBuffer result;

    for(;;) {
        if(*term == delim) {
            term++;
            DoSpaces();
            return result;
        }
        else
            if(*term == '\\') {
                switch(*++term) {
                    case 'a': result.Cat('\a'); term++; break;
                    case 'b': result.Cat('\b'); term++; break;
                    case 't': result.Cat('\t'); term++; break;
                    case 'v': result.Cat('\v'); term++; break;
                    case 'n': result.Cat('\n'); term++; break;
                    case 'r': result.Cat('\r'); term++; break;
                    case 'f': result.Cat('\f'); term++; break;
                    case 'x': {
```

```

int hex = 0;
if(IsXDigit(*++term)) {
    hex = ctoi(*term);
    if(IsXDigit(*++term)) {
        hex = 16 * hex + (*term >= 'A' ? ToUpper(*term) - 'A' + 10 : *term - '0');
        term++;
    }
}
result.Cat(hex);
break;
}
case 'u':
if(unescape) {
    int hex = 0;
    if(IsXDigit(*++term)) {
        hex = ctoi(*term);
        if(IsXDigit(*++term)) {
            hex = 16 * hex + (*term >= 'A' ? ToUpper(*term) - 'A' + 10 : *term - '0');
            if(IsXDigit(*++term)) {
                hex = 16 * hex + (*term >= 'A' ? ToUpper(*term) - 'A' + 10 : *term - '0');
                if(IsXDigit(*++term)) {
                    hex = 16 * hex + (*term >= 'A' ? ToUpper(*term) - 'A' + 10 : *term - '0');
                    term++;
                }
            }
        }
    }
}
result.Cat(WString(hex, 1).ToString());
}
else
    result.Cat(*term++);
break;
default:
if(*term >= '0' && *term <= '7') {
    int oct = *term++ - '0';
    if(*term >= '0' && *term <= '7')
        oct = 8 * oct + *term++ - '0';
    if(*term >= '0' && *term <= '7')
        oct = 8 * oct + *term++ - '0';
    result.Cat(oct);
}
else
    result.Cat(*term++);
break;
}
}
else {
    if(*term == '\\0')

```

```
    return result;  
    result.Cat(*term++);  
}  
}  
  
DoSpaces();  
return result;  
}
```

Thanks
