
Subject: Re: CParser: proposal of new functions

Posted by [Sender Ghost](#) on Wed, 16 May 2012 19:08:14 GMT

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

Hello, omari.

I think, you could do the same without changing the CParser class directly:

```
#include <Core/Core.h>
```

```
using namespace Upp;
```

```
class OmariCParser : public CParser {
public:
    bool IsId2(const char *s1, const char *s2);
    bool IsId3(const char *s1, const char *s2, const char *s3);
    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'); }

    OmariCParser(const char *ptr) : CParser(ptr) { }
    OmariCParser(const char *ptr, const char *fn, int line = 1) : CParser(ptr, fn, line) { }
};

bool OmariCParser::IsId2(const char *s1, const char *s2)
{
    if (!Id(s1)) return false;
    return IsId(s2);
}

bool OmariCParser::IsId3(const char *s1, const char *s2, const char *s3)
{
    if (!Id(s1) || !Id(s2)) return false;
    return IsId(s3);
}

bool OmariCParser::Id2(const char *s1, const char *s2)
{
    if (!Id(s1) || !Id(s2)) return false;
    return true;
}

bool OmariCParser::Id3(const char *s1, const char *s2, const char *s3)
{
    if (!Id(s1) || !Id(s2) || !Id(s3)) return false;
}
```

```

    return true;
}

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

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

    return ret;
}

String OmariCParser::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 OmariCParser::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;
}

CONSOLE_APP_MAIN
{
    const String filePath = "TestCase.txt";
    String text = LoadFile(filePath);

    if (text.IsVoid()) {
        SetExitCode(1);
        Cerr() << "Unable to load " << filePath << "\n";
        return;
    }

    OmariCParser p(text);
    try {
        // Parsing of text here
    }
    catch (OmariCParser::Error e) {
        Cerr() << "ERROR: " << e << '\n';
    }
}

```

The good example is Esc parser, based on CParser.
