

## convert.cm

```
/*  
  
    Copyright (c) 2012–2016 Seppo Laakko  
    http://sourceforge.net/projects/cmajor/  
  
    Distributed under the GNU General Public License, version 3 (GPLv3).  
    (See accompanying LICENSE.txt or http://www.gnu.org/licenses/gpl.html  
    )  
  
*/  
  
// Copyright (c) 1994  
// Hewlett-Packard Company  
// Copyright (c) 1996  
// Silicon Graphics Computer Systems, Inc.  
// Copyright (c) 2009 Alexander Stepanov and Paul McJones  
  
using System.Concepts;  
  
namespace System  
{  
    public class ConversionException : Exception  
    {  
        public ConversionException(const string& message_) : base(message_  
        )  
        {  
        }  
    }  
  
    public nothrow string ToString<I, U>(I x) where I is SignedInteger  
    and U is UnsignedInteger and ExplicitlyConvertible<I, U> and  
    ExplicitlyConvertible<U, byte>  
    {  
        string s;  
        U u = 0u;  
        bool neg = x < 0;  
        if (neg)  
        {  
            u = -cast<U>(x);  
        }  
        else  
        {  
            u = cast<U>(x);  
        }  
        do  
        {  
            s.Append(cast<char>(cast<byte>('0') + cast<byte>(u % 10u)));  
            u = u / 10u;  
        }  
    }  
}
```

```

    while (u != 0u);
    if (neg)
    {
        s.Append( '- ' );
    }
    Reverse(s.Begin(), s.End());
    return s;
}

public nothrow string ToString<U>(U x) where U is UnsignedInteger and
    ExplicitlyConvertible<U, byte>
{
    string s;
    do
    {
        s.Append( cast<char>(cast<byte>('0') + cast<byte>(x % 10u)));
        x = x / 10u;
    }
    while (x != 0u);
    Reverse(s.Begin(), s.End());
    return s;
}

public nothrow string ToString(int x)
{
    return ToString<int, uint>(x);
}

public nothrow string ToString(uint x)
{
    return ToString<uint>(x);
}

public nothrow string ToString(long x)
{
    return ToString<long, ulong>(x);
}

public nothrow string ToString(ulong x)
{
    return ToString<ulong>(x);
}

public string ToString(uhuge x)
{
    uhuge zero = 0u;
    uint ten = 10u;
    string s;
    do
    {
        Pair<uhuge, uint> p = divmod(x, ten);
        s.Append( cast<char>(cast<byte>('0') + cast<byte>(p.second)));
        x = p.first;
    }

```

```

    }
    while (x != zero);
    Reverse(s.Begin(), s.End());
    return s;
}

public nothrow string ToString(byte x)
{
    return ToString(cast<uint>(x));
}

public nothrow string ToString(sbyte x)
{
    return ToString(cast<int>(x));
}

public nothrow string ToString(short x)
{
    return ToString(cast<int>(x));
}

public nothrow string ToString(ushort x)
{
    return ToString(cast<uint>(x));
}

public nothrow string ToString(double x)
{
    return ToString(x, 15);
}

public nothrow string ToString(double x, int maxNumDecimals)
{
    string result;
    if (x < 0)
    {
        x = -x;
        result.Append('-');
    }
    result.Append(ToString(cast<int>(x)));
    double d = x - cast<int>(x);
    if (d > 0)
    {
        result.Append('.');
        for (int i = 0; d > 0 && i < maxNumDecimals; ++i)
        {
            d = 10 * d;
            int digit = cast<int>(d) % 10;
            result.Append(cast<char>(cast<int>('0') + digit));
            d = d - cast<int>(d);
        }
    }
    return result;
}

```

```

}

public nothrow string ToString(char c)
{
    string s(c);
    return s;
}

public nothrow string ToString(bool b)
{
    if (b)
    {
        return "true";
    }
    return "false";
}

public nothrow string ToString(Date date)
{
    string d;
    d.Append(cast<char>(cast<int>('0') + (date.Year() / 1000)));
    d.Append(cast<char>(cast<int>('0') + ((date.Year() % 1000) / 100)
    ));
    d.Append(cast<char>(cast<int>('0') + ((date.Year() % 100) / 10)));
    ;
    d.Append(cast<char>(cast<int>('0') + (date.Year() % 10)));
    d.Append('-');
    d.Append(cast<char>(cast<int>('0') + (date.Month() / 10)));
    d.Append(cast<char>(cast<int>('0') + (date.Month() % 10)));
    d.Append('-');
    d.Append(cast<char>(cast<int>('0') + (date.Day() / 10)));
    d.Append(cast<char>(cast<int>('0') + (date.Day() % 10)));
    return d;
}

public Date ParseDate(const string& s)
{
    if (s.Length() != 10)
    {
        throw ConversionException("invalid date '" + s + "'");
    }
    if (s[0] < '0' || s[0] > '9') throw ConversionException("invalid
    date '" + s + "'");
    if (s[1] < '0' || s[1] > '9') throw ConversionException("invalid
    date '" + s + "'");
    if (s[2] < '0' || s[2] > '9') throw ConversionException("invalid
    date '" + s + "'");
    if (s[3] < '0' || s[3] > '9') throw ConversionException("invalid
    date '" + s + "'");
    if (s[4] != '-') throw ConversionException("invalid date '" + s +
    "'");
    if (s[5] < '0' || s[5] > '9') throw ConversionException("invalid
    date '" + s + "'");

```

```

    if (s[6] < '0' || s[6] > '9') throw ConversionException("invalid
        date '" + s + "'");
    if (s[7] != '-' ) throw ConversionException("invalid date '" + s +
        "'");
    if (s[8] < '0' || s[8] > '9') throw ConversionException("invalid
        date '" + s + "'");
    if (s[9] < '0' || s[9] > '9') throw ConversionException("invalid
        date '" + s + "'");
    int y = (cast<int>(s[0]) - cast<int>('0')) * 1000 + (cast<int>(s
        [1]) - cast<int>('0')) * 100 + (cast<int>(s[2]) - cast<int>('0
        ')) * 10 + (cast<int>(s[3]) - cast<int>('0'));
    if (y < 1 || y > 9999) throw ConversionException("invalid date '"
        + s + "'");
    int m = (cast<int>(s[5]) - cast<int>('0')) * 10 + (cast<int>(s
        [6]) - cast<int>('0'));
    if (m < 1 || m > 12) throw ConversionException("invalid date '" +
        s + "'");
    int d = (cast<int>(s[8]) - cast<int>('0')) * 10 + (cast<int>(s
        [9]) - cast<int>('0'));
    if (d < 1 || d > 31) throw ConversionException("invalid date '" +
        s + "'");
    return Date(cast<ushort>(y), cast<byte>(m), cast<byte>(d));
}

public nothrow inline char HexChar(byte nibble)
{
    #assert((nibble & ~0x0Fu) == 0u);
    const char* hex = "0123456789ABCDEF";
    return hex[nibble];
}

public nothrow string ToHexString<U>(U x) where U is UnsignedInteger
and ExplicitlyConvertible<U, byte>
{
    string s;
    int n = cast<int>(sizeof(x));
    for (int i = 0; i < n; ++i)
    {
        byte b = cast<byte>(x & 0xFFu);
        s.Append(HexChar(b & 0x0Fu)); // note: low order nibble first
        because of reverse
        s.Append(HexChar(b >> 4u));
        x = x >> 8u;
    }
    Reverse(s.Begin(), s.End());
    return s;
}

public nothrow string ToHexString(byte b)
{
    string s;
    s.Append(HexChar(b >> 4u));
    s.Append(HexChar(b & 0x0Fu));
}

```

```

    return s;
}

public nothrow string ToHexString(ushort u)
{
    return ToHexString<ushort>(u);
}

public nothrow string ToHexString(uint u)
{
    return ToHexString<uint>(u);
}

public nothrow string ToHexString(ulong u)
{
    return ToHexString<ulong>(u);
}

public nothrow string ToHexString(uhuge x)
{
    string s;
    uhuge byteMask = 0xFFu;
    uhuge byteBits = 8u;
    int n = cast<int>(sizeof(x));
    for (int i = 0; i < n; ++i)
    {
        uhuge m = x & byteMask;
        byte b = cast<byte>(m.l);
        s.Append(HexChar(b & 0x0Fu)); // note: low order nibble first
        because of reverse
        s.Append(HexChar(b >> 4u));
        x = x >> byteBits;
    }
    Reverse(s.Begin(), s.End());
    return s;
}

public string ToUtf8(uint c)
{
    string s;
    if (c < 0x80)
    {
        s.Append(cast<char>(c & 0x7F));
    }
    else if (c < 0x800)
    {
        byte b1 = 0x80;
        for (int i = 0; i < 6; ++i)
        {
            b1 = b1 | cast<byte>((c & 1) << i);
            c = cast<uint>(c >> 1);
        }
        byte b0 = 0xC0;

```

```

    for (int i = 0; i < 5; ++i)
    {
        b0 = b0 | cast<byte>((c & 1) << i);
        c = cast<uint>(c >> 1);
    }
    s.Append(cast<char>(b0));
    s.Append(cast<char>(b1));
}
else if (c < 0x10000)
{
    byte b2 = 0x80;
    for (int i = 0; i < 6; ++i)
    {
        b2 = b2 | cast<byte>((c & 1) << i);
        c = cast<uint>(c >> 1);
    }
    byte b1 = 0x80;
    for (int i = 0; i < 6; ++i)
    {
        b1 = b1 | cast<byte>((c & 1) << i);
        c = cast<uint>(c >> 1);
    }
    byte b0 = 0xE0;
    for (int i = 0; i < 4; ++i)
    {
        b0 = b0 | cast<byte>((c & 1) << i);
        c = cast<uint>(c >> 1);
    }
    s.Append(cast<char>(b0));
    s.Append(cast<char>(b1));
    s.Append(cast<char>(b2));
}
else if (c < 0x110000)
{
    byte b3 = 0x80;
    for (int i = 0; i < 6; ++i)
    {
        b3 = b3 | cast<byte>((c & 1) << i);
        c = cast<uint>(c >> 1);
    }
    byte b2 = 0x80;
    for (int i = 0; i < 6; ++i)
    {
        b2 = b2 | cast<byte>((c & 1) << i);
        c = cast<uint>(c >> 1);
    }
    byte b1 = 0x80;
    for (int i = 0; i < 6; ++i)
    {
        b1 = b1 | cast<byte>((c & 1) << i);
        c = cast<uint>(c >> 1);
    }
    byte b0 = 0xF0;

```

```

        for (int i = 0; i < 3; ++i)
        {
            b0 = b0 | cast<byte>((c & 1) << i);
            c = cast<uint>(c >> 1);
        }
        s.Append(cast<char>(b0));
        s.Append(cast<char>(b1));
        s.Append(cast<char>(b2));
        s.Append(cast<char>(b3));
    }
    else
    {
        throw ConversionException("invalid unicode code point");
    }
    return s;
}

public bool ParseInt(const string& s, int& x)
{
    x = 0;
    if (s.IsEmpty()) return false;
    String.ConstIterator i = s.Begin();
    String.ConstIterator e = s.End();
    bool negative = false;
    if (*i == '+')
    {
        ++i;
    }
    else if (*i == '-')
    {
        negative = true;
        ++i;
    }
    if (i == e) return false;
    while (i < e)
    {
        char c = *i;
        if (c >= '0' && c <= '9')
        {
            x = 10 * x + (cast<int>(c) - cast<int>('0'));
        }
        else
        {
            return false;
        }
        ++i;
    }
    if (negative)
    {
        x = -x;
    }
    return true;
}

```



```

public int ParseInt(const string& s)
{
    int x;
    if (ParseInt(s, x))
    {
        return x;
    }
    throw ConversionException("integer value cannot be parsed from
        input string '" + s + "'");
}

public bool ParseUInt(const string& s, uint& x)
{
    x = 0u;
    if (s.IsEmpty()) return false;
    String.ConstIterator i = s.Begin();
    String.ConstIterator e = s.End();
    if (*i == '+')
    {
        ++i;
    }
    if (i == e) return false;
    while (i < e)
    {
        char c = *i;
        if (c >= '0' && c <= '9')
        {
            x = 10u * x + (cast<uint>(c) - cast<uint>('0'));
        }
        else
        {
            return false;
        }
        ++i;
    }
    return true;
}

public uint ParseUInt(const string& s)
{
    uint x;
    if (ParseUInt(s, x))
    {
        return x;
    }
    throw ConversionException("uint value cannot be parsed from input
        string '" + s + "'");
}

public bool ParseLong(const string& s, long& x)
{
    x = 0;

```

```

    if (s.IsEmpty()) return false;
    String.ConstIterator i = s.Begin();
    String.ConstIterator e = s.End();
    bool negative = false;
    if (*i == '+')
    {
        ++i;
    }
    else if (*i == '-')
    {
        negative = true;
        ++i;
    }
    if (i == e) return false;
    while (i < e)
    {
        char c = *i;
        if (c >= '0' && c <= '9')
        {
            x = 10 * x + (cast<long>(c) - cast<long>('0'));
        }
        else
        {
            return false;
        }
        ++i;
    }
    if (negative)
    {
        x = -x;
    }
    return true;
}

public long ParseLong(const string& s)
{
    long x;
    if (ParseLong(s, x))
    {
        return x;
    }
    throw ConversionException("long value cannot be parsed from input
        string '" + s + "'");
}

public bool ParseULong(const string& s, ulong& x)
{
    x = 0u;
    if (s.IsEmpty()) return false;
    String.ConstIterator i = s.Begin();
    String.ConstIterator e = s.End();
    if (*i == '+')
    {

```

```

        ++i;
    }
    if (i == e) return false;
    while (i < e)
    {
        char c = *i;
        if (c >= '0' && c <= '9')
        {
            x = 10u * x + (cast<ulong>(c) - cast<ulong>('0'));
        }
        else
        {
            return false;
        }
        ++i;
    }
    return true;
}

public ulong ParseULong(const string& s)
{
    ulong x;
    if (ParseULong(s, x))
    {
        return x;
    }
    throw ConversionException("ulong value cannot be parsed from
        input string '" + s + "'");
}

public bool ParseUHuge(const string& s, uhuge& x)
{
    uhuge zero = 0u;
    uhuge ten = 10u;
    x = zero;
    if (s.IsEmpty()) return false;
    String.ConstIterator i = s.Begin();
    String.ConstIterator e = s.End();
    if (*i == '+')
    {
        ++i;
    }
    if (i == e) return false;
    while (i < e)
    {
        char c = *i;
        if (c >= '0' && c <= '9')
        {
            uint d = cast<uint>(c) - cast<uint>('0');
            x = ten * x + uhuge(d);
        }
        else
        {

```

```

        return false;
    }
    ++i;
}
return true;
}

public uhuge ParseUHuge(const string& s)
{
    uhuge x;
    if (ParseUHuge(s, x))
    {
        return x;
    }
    throw ConversionException("uhuge value cannot be parsed from
        input string '" + s + "'");
}

public bool ParseHex(const string& s, ulong& hex)
{
    hex = 0u;
    if (s.IsEmpty()) return false;
    String.ConstIterator i = s.Begin();
    String.ConstIterator e = s.End();
    while (i < e)
    {
        char c = *i;
        if (c >= '0' && c <= '9')
        {
            hex = 16u * hex + (cast<uint>(c) - cast<uint>('0'));
        }
        else if (c >= 'A' && c <= 'F')
        {
            hex = 16u * hex + 10u + (cast<uint>(c) - cast<uint>('A'));
        }
        else if (c >= 'a' && c <= 'f')
        {
            hex = 16u * hex + 10u + (cast<uint>(c) - cast<uint>('a'));
        }
        else
        {
            return false;
        }
        ++i;
    }
    return true;
}

public ulong ParseHex(const string& s)
{
    ulong hex;

```

```

    if (ParseHex(s, hex))
    {
        return hex;
    }
    throw ConversionException("hexadecimal value cannot be parsed
        from input string '" + s + "'");
}

public bool ParseHex(const string& s, uhuge& hex)
{
    uhuge zero = 0u;
    uhuge sixteen = 16u;
    uhuge ten = 10u;
    hex = zero;
    if (s.IsEmpty()) return false;
    String.ConstIterator i = s.Begin();
    String.ConstIterator e = s.End();
    while (i < e)
    {
        char c = *i;
        if (c >= '0' && c <= '9')
        {
            hex = sixteen * hex + uhuge(cast<uint>(c) - cast<uint>('0'
                ));
        }
        else if (c >= 'A' && c <= 'F')
        {
            hex = sixteen * hex + ten + uhuge(cast<uint>(c) - cast<
                uint>('A'));
        }
        else if (c >= 'a' && c <= 'f')
        {
            hex = sixteen * hex + ten + uhuge(cast<uint>(c) - cast<
                uint>('a'));
        }
        else
        {
            return false;
        }
        ++i;
    }
    return true;
}

public uhuge ParseHexUHuge(const string& s)
{
    uhuge hex;
    if (ParseHex(s, hex))
    {
        return hex;
    }
    throw ConversionException("hexadecimal uhuge value cannot be
        parsed from input string '" + s + "'");
}

```

```

}

public bool ParseDouble(const string& s, double& x)
{
    x = 0.0;
    if (s.IsEmpty()) return false;
    String.ConstIterator i = s.Begin();
    String.ConstIterator e = s.End();
    bool negative = false;
    if (*i == '+')
    {
        ++i;
    }
    else if (*i == '-')
    {
        negative = true;
        ++i;
    }
    if (i == e) return false;
    int state = 0;
    double d = 10.0;
    int exponent = 0;
    bool negativeExponent = false;
    while (i < e)
    {
        char c = *i;
        switch (state)
        {
            case 0:
            {
                if (c >= '0' && c <= '9')
                {
                    x = 10 * x + (cast<int>(c) - cast<int>('0'));
                }
                else if (c == '.')
                {
                    state = 1;
                }
                else if (c == 'e' || c == 'E')
                {
                    state = 2;
                }
                break;
            }
            case 1:
            {
                if (c >= '0' && c <= '9')
                {
                    x = x + (cast<int>(c) - cast<int>('0')) / d;
                    d = d * 10;
                }
                else if (c == 'e' || c == 'E')
                {

```

```

        state = 2;
    }
    else
    {
        return false;
    }
    break;
}
case 2:
{
    if (c == '+')
    {
        state = 3;
    }
    else if (c == '-')
    {
        negatativeExponent = true;
        state = 3;
    }
    else if (c >= '0' && c <= '9')
    {
        exponent = cast<int>(c) - cast<int>('0');
        state = 3;
    }
    else
    {
        return false;
    }
    break;
}
case 3:
{
    if (c >= '0' && c <= '9')
    {
        exponent = 10 * exponent + (cast<int>(c) - cast<
            int>('0'));
    }
    else
    {
        return false;
    }
    break;
}
}
++i;
}
if (negative)
{
    x = -x;
}
if (exponent != 0)
{
    if (negatativeExponent)

```

```

        {
            exponent = -exponent;
        }
        x = x * pow(10, exponent);
    }
    return true;
}

public double ParseDouble(const string& s)
{
    double x;
    if (ParseDouble(s, x))
    {
        return x;
    }
    throw ConversionException("double value cannot be parsed from
        input string '" + s + "'");
}

public bool ParseBool(const string& s, bool& b)
{
    b = false;
    if (s == "true")
    {
        b = true;
        return true;
    }
    else if (s == "false")
    {
        return true;
    }
    return false;
}

public bool ParseBool(const string& s)
{
    bool b;
    if (ParseBool(s, b))
    {
        return b;
    }
    throw ConversionException("bool value cannot be parsed from input
        string '" + s + "'");
}
}

```