path.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;
using System.Collections;

namespace System.IO
{
    public class InvalidPathException: Exception
    {
        public InvalidPathException(const string& message_): base(
            message_)
        {
        }
        public default InvalidPathException(const InvalidPathException&);
    }

    public string GetCurrentWorkingDirectory()
    {
        IOBuffer buffer(4096u);
        char* wd = get_current_working_directory(cast<char*>(buffer.Mem()
            ), cast<int>(buffer.Size()));
        if (wd != null)
        {
            return Path.MakeCanonical(wd);
        }
        else
        {
            throw IOException("could not get current working directory");
        }
    }

    public string GetPathToExecutable()
    {
        IOBuffer buffer(4096u);
```

1

```
        int result = get_path_to_executable(cast<char*>(buffer.Mem()),
            cast<int>(buffer.Size()));
        if (result == -1)
        {
            string reason = strerror(get_errno());
            throw IOException("could not get path to current executable:
                " + reason);
        }
        string p = GetFullPath(cast<const char*>(buffer.Mem()));
        return p;
    }

    public nothrow bool FileExists(const string& filePath)
    {
        return file_exists(filePath.Chars()) != 0;
    }

    public nothrow bool DirectoryExists(const string& directoryPath)
    {
        return directory_exists(directoryPath.Chars()) != 0;
    }

    public nothrow bool PathExists(const string& path)
    {
        return path_exists(path.Chars()) != 0;
    }

    public bool LastWriteTimeLess(const string& filePath1, const string&
        filePath2)
    {
        int result = last_write_time_less(filePath1.Chars(), filePath2.
            Chars());
        if (result == -1)
        {
            string reason = strerror(get_errno());
            throw IOException("could not get last write time for file '"
                + filePath1 + "': " + reason);
        }
        else if (result == -2)
        {
            string reason = strerror(get_errno());
            throw IOException("could not get last write time for file '"
                + filePath2 + "': " + reason);
        }
        return result != 0;
    }

    public void CreateDirectories(const string& directoryPath)
    {
        string path = GetFullPath(directoryPath);
        List<string> directoriesToCreate;
        while (!DirectoryExists(path))
        {
```

```
            directoriesToCreate.Add(path);
            path = Path.GetParent(directoryPath);
        }
        int n = directoriesToCreate.Count();
        for (int i = n - 1; i >= 0; --i)
        {
            const string& directory = directoriesToCreate[i];
            int result = create_directory(directory.Chars());
        }
    }

    public void RemoveFile(const string& filePath)
    {
        int result = remove(filePath.Chars());
        if (result != 0)
        {
            string reason = strerror(get_errno());
            throw IOException("could not remove file '" + filePath + "':
                " + reason);
        }
    }

    public static class Path
    {
        public static string MakeCanonical(const string& path)
        {
            string result;
            char prev = ' ';
            for (char c : path)
            {
                if (c == '\\')
                {
                    c = '/';
                }
                if (c == '/')
                {
                    if (prev != '/')
                    {
                        result.Append(c);
                    }
                }
                else
                {
                    result.Append(c);
                }
                prev = c;
            }
            if (result.Length() == 3 && IsAlpha(result[0]) && result[1]
                == ':' && result[2] == '/')
            {
                return result;
            }
            if (result == "/")
```

```
            {
                return result;
            }
            if (!result.IsEmpty())
            {
                if (result[result.Length() - 1] == '/')
                {
                    result.Substring(0, result.Length() - 1);
                }
            }
            return result;
        }
        public static nothrow string ChangeExtension(const string& path,
            const string& extension)
        {
            int lastDotPos = path.RFind('.');
            if (extension.IsEmpty())
            {
                if (lastDotPos != -1)
                {
                    return path.Substring(0, lastDotPos);
                }
                else
                {
                    return path;
                }
            }
            else
            {
                if (lastDotPos == -1)
                {
                    #assert(!extension.IsEmpty());
                    if (extension[0] == '.')
                    {
                        return path + extension;
                    }
                    else
                    {
                        return path + "." + extension;
                    }
                }
                else
                {
                    if (extension[0] == '.')
                    {
                        return path.Substring(0, lastDotPos) + extension;
                    }
                    else
                    {
                        return path.Substring(0, lastDotPos + 1) +
                            extension;
                    }
                }
```

```
        }
    }
    public static bool HasExtension(const string& path)
    {
        string p = MakeCanonical(path);
        int lastDotPos = p.RFind('.');
        if (lastDotPos != -1)
        {
            int lastColon = p.Find(':', lastDotPos + 1);
            int lastDirSep = p.Find('/', lastDotPos + 1);
            if (lastColon > lastDotPos || lastDirSep > lastDotPos)
            {
                return false;
            }
            else if (lastDotPos < p.Length() - 1)
            {
                return true;
            }
            else
            {
                return false;
            }
        }
        else
        {
            return false;
        }
    }
    public static string GetExtension(const string& path)
    {
        string p = MakeCanonical(path);
        int lastDotPos = p.RFind('.');
        if (lastDotPos != -1)
        {
            if (p.Find('/', lastDotPos + 1) != -1)
            {
                return string();
            }
            else
            {
                return p.Substring(lastDotPos);
            }
        }
        else
        {
            return string();
        }
    }
    public static string GetFileName(const string& path)
    {
        if (path.IsEmpty())
        {
            return string();
```

```
        }
        else
        {
            string p = MakeCanonical(path);
            char lastChar = p[p.Length() - 1];
            if (lastChar == '/' || lastChar == ':')
            {
                return string();
            }
            else
            {
                int lastDirSepPos = p.RFind('/');
                if (lastDirSepPos != -1)
                {
                    return p.Substring(lastDirSepPos + 1);
                }
                else
                {
                    return p;
                }
            }
        }
    }
    public static string GetFileNameWithoutExtension(const string&
        path)
    {
        string fileName = GetFileName(path);
        int lastDotPos = fileName.RFind('.');
        if (lastDotPos != -1)
        {
            return fileName.Substring(0, lastDotPos);
        }
        else
        {
            return fileName;
        }
    }
    public static string GetDirectoryName(const string& path)
    {
        string p = MakeCanonical(path);
        if (p.IsEmpty())
        {
            return string();
        }
        else if (p.Length() == 3 && IsAlpha(p[0]) && p[1] == ':' && p
            [2] == '/')
        {
            return string();
        }
        else
        {
            int lastDirSepPos = p.RFind('/');
            if (lastDirSepPos != -1)
```

```
                {
                    return p.Substring(0, lastDirSepPos);
                }
                else
                {
                    return string();
                }
            }
        }
    }
    public static string Combine(const string& path1, const string&
        path2)
    {
        string p1 = MakeCanonical(path1);
        string p2 = MakeCanonical(path2);
        if (p1.IsEmpty())
        {
            return p2;
        }
        else if (p2.IsEmpty())
        {
            return p1;
        }
        else
        {
            if (IsAbsolute(p2))
            {
                return p2;
            }
            else
            {
                string result = p1;
                #assert(!result.IsEmpty());
                if (result[result.Length() - 1] != '/')
                {
                    result.Append('/');
                }
                result.Append(p2);
                return result;
            }
        }
    }
    public static bool IsAbsolute(const string& path)
    {
        if (path.IsEmpty())
        {
            return false;
        }
        else
        {
            string p = MakeCanonical(path);
            if (p[0] == '/')
            {
                return true;
```

```
            }
            else if (IsAlpha(p[0]) && p.Length() > 2 && p[1] == ':'
                && p[2] == '/')
            {
                return true;
            }
            else
            {
                return false;
            }
        }
    }
    public static bool IsRelative(const string& path)
    {
        return !IsAbsolute(path);
    }
    public static string GetParent(const string& path)
    {
        string fullPath = GetFullPath(path);
        int lastSlashPos = fullPath.RFind('/');
        return fullPath.Substring(0, lastSlashPos);
    }
}

public string GetFullPath(const string& path)
{
    string cp = Path.MakeCanonical(path);
    string p = cp;
    if (Path.IsRelative(p))
    {
        p = GetCurrentWorkingDirectory();
        p.Append('/');
        p.Append(cp);
    }
    List<string> components = p.Split('/');
    int w = 0;
    int n = components.Count();
    for (int i = 0; i < n; ++i)
    {
        const string& c = components[i];
        if (i == 0 || (!c.IsEmpty() && c != "."))
        {
            if (c == "..")
            {
                --w;
                if (w < 0)
                {
                    throw InvalidPathException("path '" + path + "'
                        is invalid");
                }
            }
            else
            {
```

```
                if (w != i)
                {
                    components[w] = components[i];
                }
                ++w;
            }
        }
    }
    if (w == 0)
    {
        return "/";
    }
    else if (w == 1)
    {
        const string& p = components[0];
        if (p.Length() == 2 && IsAlpha(p[0]) && p[1] == ':')
        {
            return p + "/";
        }
    }
    string result;
    for (int i = 0; i < w; ++i)
    {
        if (i != 0)
        {
            result.Append('/');
        }
        result.Append(components[i]);
    }
    return result;
    }
}
```