1 libgit2 diff 08283cbdb857d09f8e623c5c23abcaa499b6b3fc

1.1 src/common.h

```
...  ...  @@ -29,9 +29,10 @@
 29   29   # include "win32/msvc-compat.h"
 30   30   # include "win32/mingw-compat.h"
 31   31   # include "win32/error.h"
      32  +# include "win32/version.h"
 32   33   # ifdef GIT_THREADS
 33   34   #    include "win32/pthread.h"
 34       -#endif
      35  +# endif
 35   36
 36   37   #else
 37   38
```

1.2 src/transports/winhttp.c

```
...  ...  @@ -58,6 +58,7 @@ typedef struct {
 58   58       const char *service_url;
 59   59       const wchar_t *verb;
 60   60       HINTERNET request;
      61  +    wchar_t *request_uri;
 61   62       char *chunk_buffer;
 62   63       unsigned chunk_buffer_len;
 63   64       HANDLE post_body;
...  ...  @@ -145,10 +146,10 @@ static int winhttp_stream_connect(winhttp_stream *s)
145  146       winhttp_subtransport *t = OWNING_SUBTRANSPORT(s);
146  147       git_buf buf = GIT_BUF_INIT;
147  148       char *proxy_url = NULL;
148       -    wchar_t url[GIT_WIN_PATH], ct[MAX_CONTENT_TYPE_LEN];
     149  +    wchar_t ct[MAX_CONTENT_TYPE_LEN];
149  150       wchar_t *types[] = { L"*/*", NULL };
150  151       BOOL peerdist = FALSE;
151       -    int error = -1;
     152  +    int error = -1, wide_len;
152  153
153  154       /* Prepare URL */
154  155       git_buf_printf(&buf, "%s%s", t->path, s->service_url);
...  ...  @@ -156,13 +157,31 @@ static int winhttp_stream_connect(winhttp_stream *s)
156  157       if (git_buf_oom(&buf))
157  158           return -1;
158  159
159       -    git__utf8_to_16(url, GIT_WIN_PATH, git_buf_cstr(&buf));
     160  +    /* Convert URL to wide characters */
     161  +    wide_len = MultiByteToWideChar(CP_UTF8,    MB_ERR_INVALID_CHARS,
     162  +        git_buf_cstr(&buf),    -1, NULL, 0);
     163  +
     164  +    if (!wide_len) {
     165  +        giterr_set(GITERR_OS, "Failed to measure string for wide
         conversion");
     166  +        goto on_error;
     167  +    }
     168  +
     169  +    s->request_uri = git__malloc(wide_len * sizeof(wchar_t));
     170  +
     171  +    if (!s->request_uri)
     172  +        goto on_error;
     173  +
     174  +    if (!MultiByteToWideChar(CP_UTF8, MB_ERR_INVALID_CHARS,
     175  +        git_buf_cstr(&buf), -1, s->request_uri, wide_len)) {
     176  +        giterr_set(GITERR_OS, "Failed to convert string to wide form");
     177  +        goto on_error;
     178  +    }
160  179
161  180       /* Establish request */
162  181       s->request = WinHttpOpenRequest(
163  182               t->connection,
164  183               s->verb,
```

```
165     -            url,
    184 +            s->request_uri,
166 185              NULL,
167 186              WINHTTP_NO_REFERER,
168 187              types,
... ... @@ -179,19 +198,36 @@ static int winhttp_stream_connect(winhttp_stream *s)
179 198
180 199        if (proxy_url) {
181 200            WINHTTP_PROXY_INFO proxy_info;
182     -          size_t wide_len;
    201 +          wchar_t *proxy_wide;
    202 +
    203 +          /* Convert URL to wide characters */
    204 +          wide_len = MultiByteToWideChar(CP_UTF8,    MB_ERR_INVALID_CHARS,
    205 +              proxy_url, -1, NULL, 0);
183 206
184     -          git__utf8_to_16(url, GIT_WIN_PATH, proxy_url);
    207 +          if (!wide_len) {
    208 +              giterr_set(GITERR_OS, "Failed to measure string for wide
conversion");
    209 +              goto on_error;
    210 +          }
185 211
186     -          wide_len = wcslen(url);
    212 +          proxy_wide = git__malloc(wide_len * sizeof(wchar_t));
    213 +
    214 +          if (!proxy_wide)
    215 +              goto on_error;
    216 +
    217 +          if (!MultiByteToWideChar(CP_UTF8, MB_ERR_INVALID_CHARS,
    218 +              proxy_url, -1, proxy_wide, wide_len)) {
    219 +              giterr_set(GITERR_OS, "Failed to convert string to wide form");
    220 +              git__free(proxy_wide);
    221 +              goto on_error;
    222 +          }
187 223
188 224            /* Strip any trailing forward slash on the proxy URL;
189 225             * WinHTTP doesn't like it if one is present */
190     -          if (L'/' == url[wide_len - 1])
191     -              url[wide_len - 1] = L'\0';
    226 +          if (wide_len > 1 && L'/' == proxy_wide[wide_len - 2])
    227 +              proxy_wide[wide_len - 2] = L'\0';
192 228
193 229            proxy_info.dwAccessType = WINHTTP_ACCESS_TYPE_NAMED_PROXY;
194     -          proxy_info.lpszProxy = url;
    230 +          proxy_info.lpszProxy = proxy_wide;
195 231            proxy_info.lpszProxyBypass = NULL;
196 232
197 233            if (!WinHttpSetOption(s->request,
... ... @@ -199,8 +235,11 @@ static int winhttp_stream_connect(winhttp_stream *s)
199 235                &proxy_info,
200 236                sizeof(WINHTTP_PROXY_INFO))) {
201 237                giterr_set(GITERR_OS, "Failed to set proxy");
    238 +              git__free(proxy_wide);
202 239                goto on_error;
203 240            }
    241 +
    242 +          git__free(proxy_wide);
204 243        }
205 244
206 245        /* Strip unwanted headers (X-P2P-PeerDist, X-P2P-PeerDistEx) that
WinHTTP
... ... @@ -348,8 +387,15 @@ static int winhttp_stream_read(
348 387        winhttp_stream *s = (winhttp_stream *)stream;
349 388        winhttp_subtransport *t = OWNING_SUBTRANSPORT(s);
350 389        DWORD dw_bytes_read;
    390 +      char replay_count = 0;
351 391
352 392     replay:
    393 +      /* Enforce a reasonable cap on the number of replays */
    394 +      if (++replay_count >= 7) {
    395 +          giterr_set(GITERR_NET, "Too many redirects or authentication
replays");
    396 +          return -1;
    397 +      }
```

```
    398 +
353 399      /* Connect if necessary */
354 400      if (!s->request && winhttp_stream_connect(s) < 0)
355 401          return -1;
... ... @@ -445,10 +491,70 @@ replay:
445 491              WINHTTP_HEADER_NAME_BY_INDEX,
446 492              &status_code, &status_code_length,
447 493              WINHTTP_NO_HEADER_INDEX)) {
448     -                giterr_set(GITERR_OS, "Failed to retreive status code");
    494 +                giterr_set(GITERR_OS, "Failed to retrieve status code");
449 495                  return -1;
450 496          }
451 497
    498 +        /* The implementation of WinHTTP prior to Windows 7 will not
    499 +         * redirect to an identical URI. Some Git hosters use
        self-redirects
    500 +         * as part of their DoS mitigation strategy. Check first to see if
        we
    501 +         * have a redirect status code, and that we haven't already
        streamed
    502 +         * a post body. (We can't replay a streamed POST.) */
    503 +        if (!s->chunked &&
    504 +            (HTTP_STATUS_MOVED == status_code ||
    505 +             HTTP_STATUS_REDIRECT == status_code ||
    506 +             (HTTP_STATUS_REDIRECT_METHOD == status_code &&
    507 +              get_verb == s->verb) ||
    508 +             HTTP_STATUS_REDIRECT_KEEP_VERB == status_code)) {
    509 +
    510 +            /* Check for Windows 7. This workaround is only necessary on
    511 +             * Windows Vista and earlier. Windows 7 is version 6.1. */
    512 +            if (!git_has_win32_version(6, 1)) {
    513 +                wchar_t *location;
    514 +                DWORD location_length;
    515 +                int redirect_cmp;
    516 +
    517 +                /* OK, fetch the Location header from the redirect. */
    518 +                if (WinHttpQueryHeaders(s->request,
    519 +                    WINHTTP_QUERY_LOCATION,
    520 +                    WINHTTP_HEADER_NAME_BY_INDEX,
    521 +                    WINHTTP_NO_OUTPUT_BUFFER,
    522 +                    &location_length,
    523 +                    WINHTTP_NO_HEADER_INDEX) ||
    524 +                    GetLastError() != ERROR_INSUFFICIENT_BUFFER) {
    525 +                    giterr_set(GITERR_OS, "Failed to read Location header");
    526 +                    return -1;
    527 +                }
    528 +
    529 +                location = git__malloc(location_length);
    530 +                GITERR_CHECK_ALLOC(location);
    531 +
    532 +                if (!WinHttpQueryHeaders(s->request,
    533 +                    WINHTTP_QUERY_LOCATION,
    534 +                    WINHTTP_HEADER_NAME_BY_INDEX,
    535 +                    location,
    536 +                    &location_length,
    537 +                    WINHTTP_NO_HEADER_INDEX)) {
    538 +                    giterr_set(GITERR_OS, "Failed to read Location header");
    539 +                    git__free(location);
    540 +                    return -1;
    541 +                }
    542 +
    543 +                /* Compare the Location header with the request URI */
    544 +                redirect_cmp = wcscmp(location, s->request_uri);
    545 +                git__free(location);
    546 +
    547 +                if (!redirect_cmp) {
    548 +                    /* Replay the request */
    549 +                    WinHttpCloseHandle(s->request);
    550 +                    s->request = NULL;
    551 +                    s->sent_request = 0;
    552 +
    553 +                    goto replay;
    554 +                }
    555 +            }
    556 +        }
```

```
      557 +
452   558           /* Handle authentication failures */
453   559           if (HTTP_STATUS_DENIED == status_code &&
454   560               get_verb == s->verb && t->owner->cred_acquire_cb) {
...   ... @@ -747,6 +853,11 @@ static void winhttp_stream_free(git_smart_subtransport_stream
      *stream)
747   853           s->post_body = NULL;
748   854       }
749   855
      856 +     if (s->request_uri) {
      857 +         git__free(s->request_uri);
      858 +         s->request_uri = NULL;
      859 +     }
      860 +
750   861       if (s->request) {
751   862           WinHttpCloseHandle(s->request);
752   863           s->request = NULL;
...   ... @@ -876,7 +987,7 @@ static int winhttp_receivepack(
876   987  {
877   988       /* WinHTTP only supports Transfer-Encoding: chunked
878   989        * on Windows Vista (NT 6.0) and higher. */
879       -    s->chunked = LOBYTE(LOWORD(GetVersion())) >= 6;
      990 +    s->chunked = git_has_win32_version(6, 0);
880   991
881   992       if (s->chunked)
882   993           s->parent.write = winhttp_stream_write_chunked;
```

1.3 src/win32/error.c

```
...   ... @@ -8,34 +8,70 @@
8     8   #include "common.h"
9     9   #include "error.h"
10    10
      11  +#ifdef GIT_WINHTTP
      12  +# include <winhttp.h>
      13  +#endif
      14  +
      15  +#define WC_ERR_INVALID_CHARS    0x80
      16  +
11    17  char *git_win32_get_error_message(DWORD error_code)
12    18  {
13    19      LPWSTR lpMsgBuf = NULL;
      20  +    HMODULE hModule = NULL;
      21  +    char *utf8_msg = NULL;
      22  +    int utf8_size;
      23  +    DWORD dwFlags =
      24  +        FORMAT_MESSAGE_ALLOCATE_BUFFER | FORMAT_MESSAGE_IGNORE_INSERTS;
14    25
15    26      if (!error_code)
16    27          return NULL;
17    28
18        -    if (FormatMessageW(
19        -                FORMAT_MESSAGE_ALLOCATE_BUFFER |
20        -                FORMAT_MESSAGE_FROM_SYSTEM |
21        -                FORMAT_MESSAGE_IGNORE_INSERTS,
22        -                NULL, error_code, MAKELANGID(LANG_NEUTRAL, SUBLANG_DEFAULT),
23        -                (LPWSTR)&lpMsgBuf, 0, NULL)) {
24        -        int utf8_size = WideCharToMultiByte(CP_UTF8, 0, lpMsgBuf, -1, NULL,
          0, NULL, NULL);
25        -
26        -        char *lpMsgBuf_utf8 = git__malloc(utf8_size * sizeof(char));
27        -        if (lpMsgBuf_utf8 == NULL) {
28        -            LocalFree(lpMsgBuf);
29        -            return NULL;
      29  +#ifdef GIT_WINHTTP
      30  +    /* Errors raised by WinHTTP are not in the system resource table */
      31  +    if (error_code >= WINHTTP_ERROR_BASE &&
      32  +        error_code <= WINHTTP_ERROR_LAST)
      33  +        hModule = GetModuleHandleW(L"winhttp");
      34  +#endif
      35  +
      36  +    GIT_UNUSED(hModule);
```

```
      37 +
      38 +    if (hModule)
      39 +        dwFlags |= FORMAT_MESSAGE_FROM_HMODULE;
      40 +    else
      41 +        dwFlags |= FORMAT_MESSAGE_FROM_SYSTEM;
      42 +
      43 +    if (FormatMessageW(dwFlags, hModule, error_code,
      44 +        MAKELANGID(LANG_NEUTRAL, SUBLANG_DEFAULT),
      45 +        (LPWSTR)&lpMsgBuf, 0, NULL)) {
      46 +
      47 +        /* Invalid code point check supported on Vista+ only */
      48 +        if (git_has_win32_version(6, 0))
      49 +            dwFlags = WC_ERR_INVALID_CHARS;
      50 +        else
      51 +            dwFlags = 0;
      52 +
      53 +        utf8_size = WideCharToMultiByte(CP_UTF8, dwFlags,
      54 +            lpMsgBuf, -1, NULL, 0, NULL, NULL);
      55 +
      56 +        if (!utf8_size) {
      57 +            assert(0);
      58 +            goto on_error;
30    59          }
31        -        if (!WideCharToMultiByte(CP_UTF8, 0, lpMsgBuf, -1, lpMsgBuf_utf8,
          utf8_size, NULL, NULL)) {
32        -            LocalFree(lpMsgBuf);
33        -            git__free(lpMsgBuf_utf8);
34        -            return NULL;
      60 +
      61 +        utf8_msg = git__malloc(utf8_size);
      62 +
      63 +        if (!utf8_msg)
      64 +            goto on_error;
      65 +
      66 +        if (!WideCharToMultiByte(CP_UTF8, dwFlags,
      67 +            lpMsgBuf, -1, utf8_msg, utf8_size, NULL, NULL)) {
      68 +            git__free(utf8_msg);
      69 +            goto on_error;
35    70          }
36    71
      72 +on_error:
37    73          LocalFree(lpMsgBuf);
38        -        return lpMsgBuf_utf8;
39    74      }
40        -    return NULL;
      75 +
      76 +    return utf8_msg;
41    77  }
```

1.4 src/win32/version.h

```
...   ... @@ -0,0 +1,20 @@
       1 +/*
       2 + * Copyright (C) the libgit2 contributors. All rights reserved.
       3 + *
       4 + * This file is part of libgit2, distributed under the GNU GPL v2 with
       5 + * a Linking Exception. For full terms see the included COPYING file.
       6 + */
       7 +#ifndef INCLUDE_win32_version_h__
       8 +#define INCLUDE_win32_version_h__
       9 +
      10 +#include <windows.h>
      11 +
      12 +GIT_INLINE(int) git_has_win32_version(int major, int minor)
      13 +{
      14 +    WORD wVersion = LOWORD(GetVersion());
      15 +
      16 +    return LOBYTE(wVersion) > major ||
      17 +        (LOBYTE(wVersion) == major && HIBYTE(wVersion) >= minor);
      18 +}
      19 +
      20 +#endif
```