z/VSE HTTP Client Interface Description

Since VSE/ESA 2.7 there exists a HTTP Client implementation that enables you to send HTTP requests to HTTP servers outside of z/VSE. This can be used for example to retrieve HTML documents from Web servers, or call CGIs (Common Gateway Interface) or Servlets. The HTTP Client was introduced as part of the SOAP (Simple Object Access Protocol) implementation, but can also be used separately in user applications.

Overview

The z/VSE HTTP Client can be used in a CICS application as well as in a batch application. Since it is implemented in LE/C, it needs to be called from a LE conforming program or the LE environment must be set up prior to calling the HTTP Client. In CICS you do not need to take care about that, since CICS does this for you (EXEC CICS LINK).

The HTTP Client supports HTTP 1.1 with methods GET and POST (refer to RFC 1945 and RFC 2616). Method GET is used to retrieve data from a HTTP server, e.g. a HTML page or even binary content. Method POST is used to transfer data to the HTTP server and retrieve a response back. POST is typically used in HTML FORMs when calling CGIs or Servlets.

The content to be received is specified as standard URL (Unified Resource Locator), for example: http://server.domain.com:80/path/document.html. A URL contains of 4 parts: protocol (http://), server hostname (server.domain.com), the port (:80) and resource including path (/path/document.html). The port specification is optional; default port for HTTP is 80. Typically a HTTP Client connects to the server, and uses the protocol to request the resource.

In case the HTTP server is located outside of the company's intranet, a proxy or socks server may have to be used to connect to the server. The z/VSE HTTP Client implementation supports HTTP proxies and Socks (V4 and V5). This enables the HTTP Client to connect to a HTTP server through a firewall.

Keep alive mechanisms are not supported. That means the HTTP Client establishes a separate connection to the server for each HTTP request. After completion of the request the connection is disconnected.

Transfer encoding type chunked is not supported. In case a response uses "Transfer-Encoding: chunked", the HHTP client would return a non-zero return code.

Programming Interface

To use the HTTP Client a simple call has to be made from the user application to the HTTP Client. The HTTP Client is given by two programs:

- IESHTTPB – batch part
- IESHTTPC – CICS part

In batch the phase IESHTTPB has to be fetched and a direct call to the entry point is to be performed. In CICS a call is made by EXEC CICS LINK to program IESHTTPC. In both cases a parameter area is passed to the HTTP Client containing all necessary parameters. In batch, a pointer to the parameter area is passed as argument, in CICS the parameter area is given by the COMMAREA. A Return code is returned in R15 for batch, and in RESP2 for CICS.

A typical code fragment for batch in C would look as follows:

```
#include "IESHTTPH.h"
typedef int (*HTTPClient)(LPHTTP_REQ lpHttpReq);
{
  HTTPClient entrypoint;
   HTTP_REQ req;

   // setup the request block (not shown here)

  // Fetch the phase
  entrypoint = (HTTPClient)fetch("IESHTTPB");
  // call the HTTP client
  rc = entrypoint(&req);
  // release the phase
  release((void(*)())entrypoint)

  if (rc!=0)
    ...
};
```

A typical code fragment for CICS in C would look as follows:

```
#include "IESHTTPH.h"
{
   HTTP_REQ req;

   // setup the request block (not shown here)

  // call the HTTP Client
  EXEC CICS LINK PROGRAM("IESHTTPC")
                 COMMAREA(&req) LENGTH(sizeof(req))
                 RESP(rc) RESP2(rc2);
  if (rc!=DFHRESP(NORMAL))
    ...
}
```

The interface definitions are contained in the C header file IESHTTPH.h which is available in PRD1.BASE. For definitions of this area in other programming languages, please see Appendix A-C.

The Parameter area (in the code referred as HTTP_REQ) is defined as follows (C code):

```
typedef _Packed struct   HTTP_REQ
        {
          int             dwAreaLen;   // length of this control block
          // request information
          char*           szUrl;       // The URL
          unsigned int    dwUrlLen;    // Length of URL
          int             iRequest;    // Request (GET/POST)
          char*           szUserAgent; // name of user agent (or NULL)
          unsigned int    dwUserAgentLen; // Length of User Agent
          char*           szAccept;   // accept spec (or NULL)
          unsigned int    dwAcceptLen; // Length of Accept
          // user status data
          void*           lpUserData;  // user private data
          // input for POST processing
          int             iPostHandler; // handler type for post
          void*           lpPostData;   // buffer/function/program name
          unsigned int    dwPostLength; // Length of post data
          char*           szPostContentType; // content type for post
          unsigned int    dwPostContentTypeLen; // Length of cont.type
          // output for GET/POST processing
          int             iHandler; // handler type for get
          void*           lpData;   // to buffer/function/program name
          unsigned int    dwLength; // Length of get data (output)
          char*           szContentType; // content type (output)
          unsigned int    dwMaxContentTypeLen; // length of contenttype
          char*           szRetCode;   // retcd of request (output)
          unsigned int    dwMaxRetCodeLen; // length
          // proxy/socks
          int             iProxyType;  // Type of Proxy
          char*           szProxy;     // hostname/ip address of proxy
          unsigned int    dwProxyLen;  // length of proxy
          unsigned int    dwProxyPort; // port of Proxy
          char*           szUser;      // user for socks
          unsigned int    dwUserLen;   // elngth of user
          char*           szPassword;  // Password for socks
          unsigned int    dwPasswordLen; // Length of password
          // codepage settings
          char*           szAsciiCP;    // ascii code page (or NULL)
          unsigned int    dwAsciiCPLen;
          char*           szEbcdicCP;    // ebcdic code page (or NULL)
          unsigned int    dwEbcdicCPLen;
          // Additional Header Line
          char*           szHdrLine;    // Optional HTTP Header Line
          unsigned int    dwHdrLineLen;
          // New location (for 301/302 response)
          char*           szNewLocation;// Url of new location
          unsigned int    dwNewLocLen;
          // SSL settings for HTTPs
          char*           szKeyringLib; // SSL keyring libraray
          unsigned int    dwKeyringLibLen;
          char*           szKeyName;    // Name of key
          unsigned int    dwKeyNameLen;
          char*           szCiperSpecs; // SSL ciper specs
          unsigned int    dwCiperSpecsLen;
          unsigned int    dwSessionTimeout; // SSL session timeout
          // HTTP authentication
          char*           szAuthUser;   // user id of HTTP auth
          unsigned int    dwAuthUserLen;
          char*           szAuthPwd;    // password of HTTP auth
```

```
          unsigned int    dwAuthPwdLen;
          // additional SSL settings for HTTPs
          char*           szSSLType;    // SSL type: "SSL30","TLS31"
          unsigned int    dwSSLTypeLen;
        }
        HTTP_REQ;
typedef HTTP_REQ*  LPHTTP_REQ;

// Request types
#define HTTP_GET              0x00000001
#define HTTP_POST             0x00000002
#define HTTP_GET_BINARY       0x00000003 // GET, no codepage transl.
#define HTTP_POST_BINARY      0x00000004 // POST, no codepage transl.
#define HTTP_GET_TEXT         0x00000005 // GET, force codepage transl.
#define HTTP_POST_TEXT        0x00000006 // POST, force codepage transl.

// To activate trace, or the following flags into iRequest:
#define HTTP_TRACE_SYSLOG     0x01000000
#define HTTP_TRACE_SYSLST     0x02000000

// Handler types
#define HTTP_HANDLER_NOTHING   0 // no handler
#define HTTP_HANDLER_BUFFER    1 // user supplied buffer
#define HTTP_HANDLER_FUNCTION  2 // callback function via BALR
#define HTTP_HANDLER_PROGRAM   3 // callback program via LINK (CICS)

// Proxy types
#define HTTP_TYPE_DIRECT      0 // direct connection
#define HTTP_TYPE_PROXY       1 // connection through a proxy
#define HTTP_TYPE_SOCKS4      2 // connection through Socks V4
#define HTTP_TYPE_SOCKS5      3 // connection through Socks V5
```

The parameter area contains all necessary parameters to do a HTTP request. For all parameters that take string values (e.g. szURL) you must specify the address of the area containing the string and its length. This is to avoid limiting a string to a certain amount of characters.

- URL (e.g. http://www.ibm.com)
- HTTP request (GET or POST) specified as numeric constant
- user agent name (optional)
- list of accepted content type (optional)
- Information about the proxy or socks server to be used (type of proxy, proxy/socks server name, user id and password)
- Codepages to do the ASCII/EBCDIC translation
- Input and output buffers or handlers that are called to process input and output (see chapter below).
- Content type (returned)
- Return Code (returned)

Some of the parameters are optional. In case the parameter is not specified, the appropriate address must be set to NULL. Default values apply if a parameter is not specified. A return code is passed back to the caller to indicate errors during HTTP processing. For a direct call the return code is returned (R15), for CICS it is contained in RESP2:

```
#define HTTPERR_NO_ERROR          0   // NO Error
#define HTTPERR_INVALID_PARAM     1   // invalid param (e.g. NULL)
#define HTTPERR_NULL_POINTER      2   // unexpected null pointer
#define HTTPERR_NETWORK_ERR       3   // network error
```

```
#define HTTPERR_URL_ERROR         4   // URL malformed
#define HTTPERR_UNKNOWN_HOST      5   // unknown host
#define HTTPERR_CONNECT_FAILED    6   // connect to host failed
#define HTTPERR_CONNECTION_BROKEN 7   // connection is broken
#define HTTPERR_CONNECTION_CLOSED 8   // connection has been closed
#define HTTPERR_INVALID_RESP      9   // invalid response from server
#define HTTPERR_NOT_ALLOWED       10  // action not allowed by socks/pro
#define HTTPERR_CODEPAGE          11  // codepage not found
#define HTTPERR_SSL_INIT          12  // SSL initialization failed
#define HTTPERR_SSL_HANDSHAKE     13  // SSL handshake failed
#define HTTPERR_NO_MEMORY         14  // no more memory available
#define HTTPERR_COMMAREA_LEN      15  // incorrect commarea length
#define HTTPERR_PARAM_LEN         16  // length of a parameter is wrong
#define HTTPERR_PROGRAMM_ERROR    17  // error with callback program
```

Compatibility with older versions

To allow future enhancements, a new field dwAreaLen has been added at the front of the area. This field specifies the length of the parameter area. It allows the z/VSE HTTP Client code to detect which fields are set up by the application.

To support existing applications that are still using the old parameter area format (without the dwAreaLen field), the code also checks if the dwAreaLen field looks like a address. If so, it assumes that an old version format is used and internally translates it into the new format.

Existing (old) applications should therefore continue to work unchanged. They can only use the features and fields that have been available with the old format.

Newer applications should use the new format. Applications using the new format/features, will not work on z/VSE releases that do not support the new format/features.

Process a simple HTTP GET request

A simple example to retrieve content of http://www.ibm.com would fill the parameter area as follows (pseudo code). It is a good idea to clear the whole HTTP_REQ area with binary zeros before setting the required fields.

```
dwAreaLen            = Length of HTTP_REQ
szUrl                = "http://www.ibm.com"
dwUrlLen             = 18
iRequest             = HTTP_GET (=1)
szUserAgent          = NULL
dwUserAgentLen       = 0
szAccepts            = NULL
dwAcceptsLen         = 0
lpUserData           = NULL
iPostHandler         = 0
lpPostData           = NULL
dwPostLength         = 0
szPostContentType    = NULL
dwPostContentTypeLen = 0
iHandler             = HTTP_HANDLER_BUFFER (=1)
lpData               = Address of 1KB buffer in memory
dwLength             = 1024
szContentType        = Address of 100 byte area for content type (returned)
dwMaxContentTypeLen  = 100
```

```
szRetCode            = Address of 100 byte area for return code (returned)
dwMaxRetCodeLen      = 100
iProxyType           = HTTP_TYPE_DIRECT (=0)
szProxy              = NULL
dwProxyLen           = 0
dwProxyPort          = 0
szUser               = NULL
dwUserLen            = 0
szPassword           = NULL
dwPasswordLen        = 0
szAsciiCP            = NULL
dwAsciiCPLen         = 0
szEbcdicCP           = NULL
dwEbcdicCPLen        = 0
szHdrLine            = NULL
dwHdrLineLen         = 0
szNewLocation        = NULL
dwNewLocLen          = 0
szKeyringLib         = NULL
dwKeyringLibLen      = 0
szKeyName            = NULL
dwKeyNameLen         = 0
szCiperSpecs         = NULL
dwCiperSpecsLen      = 0
dwSessionTimeout     = 0
szAuthUser           = NULL
dwAuthUserLen        = 0
szAuthPwd            = NULL
dwAuthPwdLen         = 0
szSSLType            = NULL
dwSSLTypeLen         = 0
```

On return the following parameters have been updated:

```
dwLength             = actual length of content
lpData               = content of buffer has been modified
szContentType        = contains the actual content type (e.g.
"text/html") szRetCode = contains the HTTP return code (e.g. "200 OK")
```

The received content data is translated into ASCII if the content type starts with “text/”.

Processing a HTTP POST request

To process a HTTP GET, only the output buffer or handler is used. To process a HTTP POST, you also need to setup the input buffer or handler:

```
iPostHandler         = HTTP_HANDLER_BUFFER (=1)
lpPostData           = Address of buffer containing the data to
send dwPostLength    = 100 (length of data to post)
```

You also need to specify the content type of the data you send. If the content type is not specified the following content type is used: “application/x-www-form-urlencoded”

```
szPostContentType    =
"text/plain" dwPostContentTypeLen
= 10
```

The actual content data is translated into ASCII before sending, if the content type starts with “text/” or if the default content type “application/x-www-form-urlencoded” is used.

Note: If you specify HTTP_GET_BINARY or HTTP_POST_BINARY in field iRequest then no translation will be performed, regardless of the content type.

Specify a user agent name

In some cases you may want to specify a ser agent name. Some HTTP servers behave different for some user agents (e.g. Netscape vs. MS Internet Explorer). If you do not specify a user agent, the default name “VSE HTTPClient” is used.

```
szUserAgent          = "My http Client"
dwUserAgentLen       = 14
```

Specify content types to accept

Within a HTTP request (GET or POST) the client tells the HTTP server which content type it is able to process. The z/VSE HTTP Client per default uses the following accepts specification: "text/html,image/gif,image/jpeg,*". This means it accepts HTML documents (text/html), GIF images (image/gif) and JPG images (image/jpg) and also other content types (*). You may want to specify your own accepts specification in order to limit the accepted content types to XML for example (text/xml).

```
szAccepts            = "text/xml"
dwAcceptsLen         = 8
```

Specify an additional HTTP header line

Within a HTTP request (GET or POST) the client can send additional information in the HTTP header, for example SOAP uses this to add a SOAP-Action HTTP header line. You can add one additional HTTP header line by setting up the szHdrLine parameter:

```
szHdrLine            = „SOAP-Action: urn:myservcie#method"
dwHdrLineLen         = 33
```

Using a Proxy or Socks server

To connect from a company’s intranet to a HTTP server outside the company, you may need to establish a connection through the company’s firewall. Typically this is done by making use of a HTTP proxy or a Socks server. In both cases you establish a connection to the proxy or socks server instead of connecting to the final destination. The proxy or socks server then connect to the server outside the firewall and create a tunnel for the particular HTTP connection. The z/VSE HTTP Client supports HTTP proxy servers and socks servers (Version 4 and 5). For Socks V4 you must specify a user id, for socks V5 you must specify user id and password that authenticates you at the socks server.

To use a HTTP proxy you would setup the parameter area as follows:

```
iProxyType           = HTTP_TYPE_PROXY (=1)
szProxy              = "proxy.boeblingen.de.ibm.com"
dwProxyLen           = 27
```

```
dwProxyPort          = 80
szUser               = NULL
dwUserLen            = 0
szPassword           = NULL
dwPasswordLen        = 0
```

To use a socks V4 server you would setup the parameter area as follows:

```
iProxyType           = HTTP_TYPE_SOCKS4 (=2)
szProxy              = "socks1.server.ibm.com"
dwProxyLen           = 20
dwProxyPort          = 1080
szUser               = "hugo"
dwUserLen            = 4
szPassword           = NULL
dwPasswordLen        = 0
```

To use a socks V5 server you would setup the parameter area as follows:

```
iProxyType           = HTTP_TYPE_SOCKS5 (=3)
szProxy              = "socks1.server.ibm.com"
dwProxyLen           = 20
dwProxyPort          = 1080
szUser               = "hugo"
dwUserLen            = 4
szPassword           = „password"
dwPasswordLen        = 8
```

Using Input/Output handlers

The easiest way to send data to a server or receive the content of a document is to use buffers in memory that contains the data to be sent, or where the data is being received into. In this case you set the parameter iPostHandler or iHandler to 1 (HTTP_HANDLER_BUFFER) and specify the address of the buffer in lpPostData or lpData. The length of the buffers is specified in dwPostLength or dwLength (dwLength will contain the number of bytes received on return).

Using buffers in memory is fine if you know the amount of data that is being sent or received. For sending data (POST) you typically know how much data you want to send. However, when retrieving the content of a document you typically do not know the amount of data that will be received. Using a buffer that is too large is fine, but wastes storage. Using a buffer that is too small, will truncate the content to the length of your buffer.

To avoid this fixed length restriction, you can use input or output handlers. An input handler (POST handler) is a peace of code that is responsible for passing the data to be sent to the HTTP Client in several smaller peaces. The handler is called by the HTTP Client (callback) as long as input is available. An output handler is a peace of code that is called by the HTTP Client (callback) for several smaller chunks of received data.

A handler can be a sub routine within the user program (batch and CICS) or another CICS program. The type of handler is specified in iPostHandler or iHandler. Valid values are:

- HTTP_HANDLER_BUFFER = 1 user supplied buffer
- HTTP_HANDLER_FUNCTION = 2 callback function via direct call
- HTTP_HANDLER_PROGRAM = 3 callback program via LINK (CICS)

- HTTP_HANDLER_CONTAINER = 4 GET/PUT container (CICS)

The entry point of the handler sub routine or the CICS handler program is specified in lpPostData or lpData.

A handler sub routine is called with the address of a parameter area. In case of a CICS handler program the COMMAREA is used as parameter area:

```
// Commarea for Handler program
typedef _Packed struct   HTTP_HANDLER_PARAM
        {
          LPHTTP_REQ   lpHttpReq;  // underlying HTTP Request
          area void*   lpBuffer;   // ptr to buffer
          unsigned int dwLength;   // Length of buffer
        }
        HTTP_HANDLER_PARAM;
typedef HTTP_HANDLER_PARAM*   LPHTTP_HANDLER_PARAM;

// Handler callback function
typedef int (*HTTP_HandlerProc)(LPHTTP_HANDLER_PARAM lpParam);
```

The parameter area contains:

- The address of the HTTP Request parameter block (as back reference)
- The address of a buffer
- The length of the buffer

The handler is responsible for filling the buffer (POST) up to dwLength bytes and set dwLength to the actual amount of bytes, or processing the data contained in the buffer (GET). The handler is sequentially called until no more data is to be processed. For POST the dwPostLength parameter specifies the amount of data to be processed, for data retrieval the handler is called until no more data is to be received.

The HTTP request parameter area contains a field called lpUserData. This field is not touched by the HTPP Client, but it can be used to store a pointer to any kind of user data. Since a handler also gets the HTTP parameter area within the handler parameter area (lpHttpReq), the handler can refer to the user data specified in the originating HTTP request. This allows the handler to distinguish between different HTTP requests being processed by the same handler code.

Specifying ASCII and EBCDIC code pages

The z/VSE HTTP Client uses the Language Environment (LE) codepage translator function iconv to translate textual content from ASCII to EBCDIC and vice versa. Please refer to the LE documentation for details on iconv and available codepages. Per default the ASCII codepage ISO8859-1 and the EBCDIC codepage IBM-1047 is used. You may wan to specify other codepages. For a list of codepages available please refer to C-Runtime Programming Guide, SC33-6688, Chapter: Internalization.

```
szAsciiCP           = "ISO8859-1"
dwAsciiCPLen        = 9
szEbcdicCP          = "IBM-1047"
dwEbcdicCPLen       = 8
```

The HTTP Client respects charset specifications in the content type:

```
ContentType = text/html; charset=ISO-8859-1
```

In case of a POST request, the content type given in szPostContentType is searched for a charset specification. In case of a GET request the content type received from the HTTP server is searched for a charset specification. If a charset is specified/found, the HTTP Client translates it into a LE codepage translator codepage name. For example, the HTTP charset ISO-8859-1 is translated into the ASCII codepage IBM8859-1. The HTTP Client then tries to load the appropriate codepage translators. In case the ASCII codepage can not be loaded, the codepages specified by the szAsciiCP field in the parameter block is used. The EBCDIC codepage name is always given by the szEbcdicCP field in the parameter block.

Note: If you specify HTTP_GET_BINARY or HTTP_POST_BINARY in field iRequest then no translation will be performed, regardless of the content type.

Handling URL redirects

An HTTP server may respond with HTTP status codes 301 or 302 to inform the client that the content requested has been moved permanently (301) or temporary (302). The new location is sent back in an HTTP header field. To receive the new location, you can specify a buffer that will contain the URL of the new location on return, if the server returned HTTP status code 301 or 302. Check szRetCode field for the status code. The application should then execute another request with the new URL.

```
szNewLocation        = Address of buffer for new location
dwNewLocLen          = Length of new location buffer
```

Using HTTP over SSL (HTTPS)

The z/VSE HTTP Client supports HTTP over SSL (Secure Socket Layer). HTTP over SSL is indicated by the protocol specification of “https://” in the URL. If the URL specified starts with “https://” then the HTTP Client will automatically use SSL to secure the connection.

To use SSL, you must specify some SSL related parameters, such as the Keyring Library (e.g. CRYPTO.KEYRING), the key name and the SSL cipher suites:

```
szKeyringLib         = “CRYPTO.KEYRING”
dwKeyringLibLen      = 14
szKeyName            = “SSLKEY”
dwKeyNameLen         = 6
szCiperSpecs         = “090A622F35”
dwCiperSpecsLen      = 10
dwSessionTimeout     = 86400
```

Appendix A – Parameter area definitions in Assembler

```
* PARAMETER AREA FOR HTTP CLIENT
HTTPREQ   DSECT        HTTP REQUEST DSECT
AREALEN   DS   F       LENGTH OF THIS BLOCK
URL       DS   A       ADDR OF AREA CONTAINING THE URL
URLLEN    DS   F       LENGTH OF URL
REQUEST   DS   F       REQUEST (GET/POST)
HTTPGET   EQU  1       GET REQUEST
HTTPPOST  EQU  2       POST REQUEST
HTTPGETB  EQU  3       GET BINARY REQUEST
HTTPPOSB  EQU  4        POST BINARY REQUEST
USRAGENT  DS   A       ADDR OF AREA CONT. NAME OF USER AGENT (OR NULL)
USRAGLEN  DS   F       LENGTH OF USER AGENT
ACCEPTS   DS   A       ADDR OF AREA CONT. THE ACCEPTS SPEC (OR NULL)
ACCLEN    DS   F       LENGTH OF ACCEPTS
USERDATA  DS   A       USER PRIVATE DATA
POSTHDLR  DS   F       HANDLER TYPE FOR POST
HDLRBUFF  EQU  1       USER SUPPLIED BUFFER
HDLRFUNC  EQU  2       CALLBACK FUNCTION VIA BALR
HDLRPROG  EQU  3       CALLBACK PROGRAM VIA LINK (CICS)
POSTDATA  DS   A       ADDR OF BUFFER/FUNCTION/PROGRAM NAME
POSTLEN   DS   F       LENGTH OF POST DATA
POSTCTYP  DS   A       ADDR OF AREA FOR CONTENT TYPE FOR POST
PCTYPLEN  DS   F       LENGTH OF CONT.TYPE
HANDLER   DS   F       GET-HANDLER TYPE (HDLRBUFF/HDLRFUNC/HDLRPROG)
DATA      DS   A       ADDR OF BUFFER/FUNCTION/PROGRAM NAME
DATALEN   DS   F       LENGTH OF GET DATA (INPUT/OUTPUT)
CONTTYPE  DS   A       ADDR OF AREA FOR CONTENT TYPE (OUTPUT)
MCTYPLEN  DS   F       LENGTH OF CONTENTTYPE AREA
RETCODE   DS   A       ADDR OF AREA FOR RETCD OF REQUEST (OUTPUT)
MRCODLEN  DS   F       LENGTH OF RETCODE AREA
PROXYTYP  DS   F       TYPE OF PROXY
PDIRECT   EQU  0       DIRECT CONNECTION
PPROXY    EQU  1       CONNECTION THROUGH A PROXY
PSOCKS4   EQU  2       CONNECTION THROUGH SOCKS V4
PSOCKS5   EQU  3       CONNECTION THROUGH SOCKS V5
PROXY     DS   A       ADDR OF AREA CONTAINING HOSTNAME/IP OF PROXY
PROXYLEN  DS   F       LENGTH OF PROXY
PRXYPORT  DS   F       PORT OF PROXY
USER      DS   A       USER FOR SOCKS
USERLEN   DS   F       LENGTH OF USER
PASSWORD  DS   A       PASSWORD FOR SOCKS
PASSWLEN  DS   F       LENGTH OF PASSWORD
ASCIICP   DS   A       ASCII CODE PAGE (OR NULL)
ASCIILEN  DS   F       LENGTH OF ASCII CODEPAGE
EBCDICCP  DS   A       EBCDIC CODE PAGE (OR NULL)
EBCDILEN  DS   F       LENGTH OF EBCDIC CODEPAGE
HDRLINE   DS   A       ADDITIONAL HTTP HEADER LINE (OR NULL)
HDRLNLEN  DS   F       LENGTH OF HEADER LINE
NEWLOC    DS   A       URL OF NEW LOCATION (FOR 301/302 RESPONSES)
NEWLOCLN  DS   F       LENGTH OF NEW LOCATION
KEYRNGLIB DS   A       SSL KEY-RING LIBRARY
KRNGLBLEN DS   F       LENGTH OF SSL KEY-RING LIBRARY
KEYNAME   DS   A       SSL KEY NAME
KNAMELEN  DS   F       LENGTH OF SSL KEY NAME
CIPSPECS  DS   A       SSL CIPHER SPECS
CIPSPLEN  DS   F       LENGTH OF SSL CIPHER SPECS
SESSTOUT  DS   F       SSL SESSION TIMEOUT
AUTHUSER  DS   A       USER NAME FOR HTTP AUTHENTICATION
AUSERLEN  DS   F       LENGTH OF USER NAME
```

```
AUTHPWD   DS   A       PASSWORD FOR HTTP AUTHENTICATION
APWDLEN   DS   F       LENGTH OF PASSWORD
SSLTYPE   DS   A       SSL TYPE, E.G. "SSL30"
SSLTPLEN  DS   F       LENGTH OF SSL TYPE
*
* PARAMETER AREA FOR INPUT/OUTPUT HANDLER
HDLRPARM  DSECT        PARAMETER AREA FOR HANDLER PROGRAM
HTTPREQA  DS   A       UNDERLYING HTTP REQUETS AREA
BUFFER    DS   A       ADDR OF BUFFER
LENGTH    DS   F       LENGTH OF BUFFER
*
* ERROR CODES
ENOERROR  EQU  0       NO ERROR
EINVPARM  EQU  1       INVALID PARAM (E.G. NULL)
ENULLPTR  EQU  2       UNEXPECTED NULL POINTER
ENETWORK  EQU  3       NETWORK ERROR
EURL      EQU  4       URL MALFORMED
EUNKHOST  EQU  5       UNKNOWN HOST
ECONNECT  EQU  6       CONNECT TO HOST FAILED
ECONBRKN  EQU  7       CONNECTION IS BROKEN
ECONCLSD  EQU  8       CONNECTION HAS BEEN CLOSED
EINVRESP  EQU  9       INVALID RESPONSE FROM SERVER
ENOTALLW  EQU  10      ACTION NOT ALLOWED BY SOCKS/PROXY
ECODEPGE  EQU  11      CODEPAGE NOT FOUND
ESSLINIT  EQU  12      SSL INITIALIZATION FAILED
ESSLHNDS  EQU  13      SSL HANDSHAKE HAS FAILED
ENOMEM    EQU  14      NOT ENOUGH MEMORY/GETVIS
ECALEN    EQU  15      COMMAREA LENGTH IS WRONG
EPARMLEN  EQU  16      LENGTH OF A PARAMETER IS WRONG
```

Appendix B – Parameter area definitions in PL/I

```
/* HTTP CLIENT PARAMETER AREA
*/ DCL 1 HTTP_REQ BASED,
      2 AREALEN         BIXED BIN(31)  /* LENGTH OF THE AREA */
      2 URL             POINTER,       /* THE URL */
      2 URLLEN          FIXED BIN(31), /* LENGTH OF URL */
      2 REQUEST         FIXED BIN(31), /* REQUEST (GET/POST) */
      2 USERAGENT       POINTER,       /* NAME OF USER AGENT (OR NULL) */
      2 USERAGENTLEN    FIXED BIN(31), /* LENGTH OF USER AGENT */
      2 ACCEPTS         POINTER,       /* ACCEPTS SPEC (OR NULL) */
      2 ACCEPTSLEN      FIXED BIN(31), /* LENGTH OF ACCEPTS */
      2 USERDATA        POINTER,       /* USER PRIVATE DATA */
      2 POSTHANDLER     FIXED BIN(31), /* HANDLER TYPE FOR POST */
      2 POSTDATA        POINTER,       /* BUFFER/FUNCTION/PROGRAM NAME */
      2 POSTLENGTH      FIXED BIN(31), /* LENGTH OF POST DATA */
      2 POSTCONTENTTYPE POINTER,       /* CONTENT TYPE FOR POST */
      2 PCONTENTTYPELEN FIXED BIN(31), /* LENGTH OF CONT.TYPE */
      2 HANDLER         FIXED BIN(31), /* HANDLER TYPE FOR GET */
      2 DATA            POINTER,       /* BUFFER/FUNCTION/PROG NAME */
      2 LENGTH          FIXED BIN(31), /* LENGTH OF GET DATA (OUTPUT) */
      2 CONTENTTYPE     POINTER,       /* CONTENT TYPE (OUTPUT) */
      2 CONTENTTYPELEN  FIXED BIN(31), /* LENGTH OF CONTENTTYPE */
      2 RETCODE         POINTER,       /* RETCD OF REQUEST (OUTPUT) */
      2 RETCODELEN      FIXED BIN(31), /* LENGTH OF RETCODE */
      2 PROXYTYPE       FIXED BIN(31), /* TYPE OF PROXY */
      2 PROXY           POINTER,       /* HOSTNAME/IP ADDRESS OF PROXY */
      2 PROXYLEN        FIXED BIN(31), /* LENGTH OF PROXY */
      2 PROXYPORT       FIXED BIN(31), /* PORT OF PROXY */
      2 USER            POINTER,       /* USER FOR SOCKS */
      2 USERLEN         FIXED BIN(31), /* LENGTH OF USER */
      2 PASSWORD        POINTER,       /* PASSWORD FOR SOCKS */
      2 PASSWORDLEN     FIXED BIN(31), /* LENGTH OF PASSWORD */
      2 ASCIICP         POINTER,       /* ASCII CODE PAGE (OR NULL) */
      2 ASCIICPLEN      FIXED BIN(31), /* LEN OF ASCII CP */
      2 EBCDICCP        POINTER,       /* EBCDIC CODE PAGE (OR NULL) */
      2 EBCDICCPLEN     FIXED BIN(31), /* LEN OF EBCDIC CP */
      2 HDRLINE         POINTER,       /* HTTP HEADER LINE (OR NULL) */
      2 HDRLINELEN      FIXED BIN(31), /* LEN OF HEADER LINE */
      2 NEWLOCATION     POINTER,       /* URL OF NEW LOCATION */
      2 NEWLOCACTIONLEN FIXED BIN(31), /* LENGTH OF NEW LOCATION */
      2 KEYRINGLIB      POINTER,       /* SSL KEYRING LIBRARY */
      2 KEYRINGLIBLEN   FIXED BIN(31), /* LENGTH OF SSL KEYRING LIBRARY
      */
      2 KEYNAME         POINTER,       /* SSL KEY NAME  */
      2 KEYNAMELEN      FIXED BIN(31), /* LENGTH OF SSL KEY NAME */
      2 CIPERSPECS      POINTER,       /* SSL CIPER SPECS */
      2 CIPERSPECSLEN   FIXED BIN(31), /* LENGTH OF SSL CIPER SPECS */
      2 SESSTIMEOUT     FIXED BIN(31), /* SSL SESSION TIMEOUT */
      2 AUTHUSER        POINTER,       /* USER FOR HTTP AUTHENTICATION */
      2 AUTHUSERLEN     FIXED BIN(31), /* LENGTH OF USER */
      2 AUTHPWD         POINTER,       /* PASSWD FOR HTTP AUTHENTICATION
      */
      2 AUTHPWDLEN      FIXED BIN(31), /* LENGTH OF PASSWORD */
      2 SSLTYPE         POINTER,       /* SSL TYPE, E.G. "SSL30" */
      2 SSLTYPELEN      FIXED BIN(31); /* LENGTH IOF SSL TYPE */

/* HANDLER PARAMETER AREA
*/ DCL 1 HTTP_HANDLER_PARAM
BASED
      2 HTTPREQ         POINTER,       /* UNDERLYING HTTP REQUETS AREA */
```

```
      2 BUFFER          POINTER,       /* PTR TO BUFFER */
      2 LENGTH          FIXED BIN(31); /* LENGTH OF BUFFER */

/* VALUES FOR HTTP REQUEST */
DCL HTTP_GET               FIXED BIN(31) STATIC INIT(1);
DCL HTTP_POST              FIXED BIN(31) STATIC INIT(2);
DCL HTTP_GET_BINARY        FIXED BIN(31) STATIC INIT(3);
DCL HTTP_POST_BINARY       FIXED BIN(31) STATIC INIT(4);
DCL HTTP_GET_TEXT          FIXED BIN(31) STATIC INIT(5);
DCL HTTP_POST_TEXT         FIXED BIN(31) STATIC INIT(6);

/* VALUES FOR HANDLER TYPE */
DCL  HTTP_HANDLER_BUFFER  FIXED  BIN(31)  STATIC  INIT(1);
DCL HTTP_HANDLER_FUNCTION FIXED BIN(31) STATIC INIT(2);
DCL HTTP_HANDLER_PROGRAM   FIXED BIN(31) STATIC INIT(3);

/* VALUES FOR PROXY TYPE */
DCL  HTTP_TYPE_DIRECT     FIXED  BIN(31)  STATIC  INIT(0);
DCL  HTTP_TYPE_PROXY      FIXED  BIN(31)  STATIC  INIT(1);
DCL  HTTP_TYPE_SOCKS4     FIXED  BIN(31)  STATIC  INIT(2);
DCL HTTP_TYPE_SOCKS5       FIXED BIN(31) STATIC INIT(3);

/* RETURN CODES */
DCL HTTPERR_NO_ERROR       FIXED BIN(31) STATIC INIT(0);
DCL HTTPERR_INVALID_PARAM  FIXED BIN(31) STATIC INIT(1);
DCL HTTPERR_NULL_POINTER   FIXED BIN(31) STATIC INIT(2);
DCL HTTPERR_NETWORK_ERR    FIXED BIN(31) STATIC INIT(3);
DCL HTTPERR_URL_ERROR      FIXED BIN(31) STATIC INIT(4);
DCL HTTPERR_UNKNOWN_HOST   FIXED BIN(31) STATIC INIT(5);
DCL HTTPERR_CONNECT_FAILED FIXED BIN(31) STATIC INIT(6);
DCL HTTPERR_CONNECTION_BROKEN FIXED BIN(31) STATIC INIT(7);
DCL HTTPERR_CONNECTION_CLOSED FIXED BIN(31) STATIC INIT(8);
DCL HTTPERR_INVALID_RESP   FIXED BIN(31) STATIC INIT(9);
DCL HTTPERR_NOT_ALLOWED    FIXED BIN(31) STATIC INIT(10);
DCL HTTPERR_CODEPAGE       FIXED BIN(31) STATIC INIT(11);
DCL HTTPERR_SSL_INIT       FIXED BIN(31) STATIC INIT(12);
DCL HTTPERR_SSL_HANDSHAKE  FIXED BIN(31) STATIC INIT(13);
DCL HTTPERR_NO_MEMORY      FIXED BIN(31) STATIC INIT(14);
DCL HTTPERR_COMMAREA_LEN   FIXED BIN(31) STATIC INIT(15);
DCL HTTPERR_PARAM_LEN      FIXED BIN(31) STATIC INIT(16);
```

Appendix C – Parameter area definitions in COBOL

```
*****************************
* HTTP CLIENT PARAMETER AREA *
*****************************
 01   HTTP-REQ.
      02 AREALEN          PIC 9(9) BINARY.
      02 URL              USAGE IS POINTER.
      02 URLLEN           PIC 9(9) BINARY.
      02 REQUEST          PIC 9(9) BINARY.
      02 USERAGENT        USAGE IS POINTER.
      02 USERAGENTLEN     PIC 9(9) BINARY.
      02 ACCEPTS          USAGE IS POINTER.
      02 ACCEPTSLEN       PIC 9(9) BINARY.
      02 USERDATA         USAGE IS POINTER.
      02 POSTHANDLER      PIC 9(9) BINARY.
      02 POSTDATA         USAGE IS POINTER.
      02 POSTLENGTH       PIC 9(9) BINARY.
      02 POSTCONTENTTYPE  USAGE IS POINTER.
      02 PCONTENTTYPELEN  PIC 9(9) BINARY.
      02 HANDLER          PIC 9(9) BINARY.
      02 DATA             USAGE IS POINTER.
      02 LENGTH           PIC 9(9) BINARY.
      02 CONTENTTYPE      USAGE IS POINTER.
      02 CONTENTTYPELEN   PIC 9(9) BINARY.
      02 RETCODE          USAGE IS POINTER.
      02 RETCODELEN       PIC 9(9) BINARY.
      02 PROXYTYPE        PIC 9(9) BINARY.
      02 PROXY            USAGE IS POINTER.
      02 PROXYLEN         PIC 9(9) BINARY.
      02 PROXYPORT        PIC 9(9) BINARY.
      02 USER             USAGE IS POINTER.
      02 USERLEN          PIC 9(9) BINARY.
      02 PASSWORD         USAGE IS POINTER.
      02 PASSWORDLEN      PIC 9(9) BINARY.
      02 ASCIICP          USAGE IS POINTER.
      02 ASCIICPLEN       PIC 9(9) BINARY.
      02 EBCDICCP         USAGE IS POINTER.
      02 EBCDICCPLEN      PIC 9(9) BINARY.
      02 HDRLINE          USAGE IS POINTER.
      02 HDRLINELEN       PIC 9(9) BINARY.
      02 NEWLOCATION      USAGE IS POINTER.
      02 LEWLOCATIONLEN   PIC 9(9) BINARY.
      02 KEYRINGLIB       USAGE IS POINTER.
      02 KEYRINGLIBLEN    PIC 9(9) BINARY.
      02 KEYNAME          USAGE IS POINTER.
      02 KEYNAMELEN       PIC 9(9) BINARY.
      02 CIPERSPECS       USAGE IS POINTER.
      02 CIPERSPECSLEN    PIC 9(9) BINARY.
      02 SESSTIMEOUT      PIC 9(9) BINARY.
      02 AUTHUSER         USAGE IS POINTER.
      02 AUTHUSERLEN      PIC 9(9) BINARY.
      02 AUTHPWD          USAGE IS POINTER.
      02 AUTHPWDLEN       PIC 9(9) BINARY.
      02 SSLTYPE          USAGE IS POINTER.
      02 SSLTYPELEN       PIC 9(9) BINARY.



*****************************
* HANDLER PARAMETER AREA     *
```

```
******************************
 01   HANDLER-PARM.
      02 HTTPREQ         USAGE IS POINTER.
      02 BUFFER          USAGE IS POINTER.
      02 LENGTH          PIC 9(9) BINARY.

******************************
* VALUES FOR HTTP REQUEST    *
******************************
 01 HTTP-GET             PIC 9(9) BINARY VALUE 1.
 01 HTTP-POST            PIC 9(9) BINARY VALUE 2.
 01 HTTP-GET-BINARY      PIC 9(9) BINARY VALUE 3.
 01 HTTP-POST-BINARY     PIC 9(9) BINARY VALUE 4.
 01 HTTP-GET-TEXT        PIC 9(9) BINARY VALUE 5.
 01 HTTP-POST-TEXT       PIC 9(9) BINARY VALUE 6.


******************************
* VALUES FOR HANDLER TYPE    *
******************************
 01 HTTP-HANDLER-BUFFER   PIC 9(9) BINARY VALUE 1.
 01 HTTP-HANDLER-FUNCTION PIC 9(9) BINARY VALUE 2.
 01 HTTP-HANDLER-PROGRAM  PIC 9(9) BINARY VALUE 3.

******************************
* VALUES FOR PROXY TYPE      *
******************************
 01 HTTP-TYPE-DIRECT      PIC 9(9) BINARY VALUE 0.
 01 HTTP-TYPE-PROXY       PIC 9(9) BINARY VALUE 1.
 01 HTTP-TYPE-SOCKS4      PIC 9(9) BINARY VALUE 2.
 01 HTTP-TYPE-SOCKS5      PIC 9(9) BINARY VALUE 3.

******************************
* RETURN CODES               *
******************************
 01 HTTPERR-NO-ERROR       PIC 9(9) BINARY VALUE 0.
 01 HTTPERR-INVALID-PARAM  PIC 9(9) BINARY VALUE 1.
 01 HTTPERR-NULL-POINTER   PIC 9(9) BINARY VALUE 2.
 01 HTTPERR-NETWORK-ERR    PIC 9(9) BINARY VALUE 3.
 01 HTTPERR-URL-ERROR      PIC 9(9) BINARY VALUE 4.
 01 HTTPERR-UNKNOWN-HOST   PIC 9(9) BINARY VALUE 5.
 01 HTTPERR-CONNECT-FAILED PIC 9(9) BINARY VALUE 6.
 01 HTTPERR-CONNECTION-BROKEN PIC 9(9) BINARY VALUE 7.
 01 HTTPERR-CONNECTION-CLOSED PIC 9(9) BINARY VALUE 8.
 01 HTTPERR-INVALID-RESP   PIC 9(9) BINARY VALUE 9.
 01 HTTPERR-NOT-ALLOWED    PIC 9(9) BINARY VALUE 10.
 01 HTTPERR-CODEPAGE       PIC 9(9) BINARY VALUE 11.
 01 HTTPERR-SSL-INIT       PIC 9(9) BINARY VALUE 12.
 01 HTTPERR-SSL-HANDSHAKE  PIC 9(9) BINARY VALUE 13.
 01 HTTPERR-NO-MEMORY      PIC 9(9) BINARY VALUE 14.
 01 HTTPERR-COMMAREA-LEN   PIC 9(9) BINARY VALUE 15.
 01 HTTPERR-PARAM-LEN      PIC 9(9) BINARY VALUE 16.
```

Remarks

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs.

Trademarks

IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corp., registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the web at "Copyright and trademark information" at www.ibm.com/legal/copytrade.shtml.

Linux is a registered trademark of Linus Torvalds in the United States, other countries, or both.

Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Java and all Java-based trademarks and logos are trademarks or registered trademarks of Oracle and/or its affiliates.

Comments and Questions

Comments or questions on this documentation are welcome. Please send your comments to:

zvse@de.ibm.com