ISAPI-Interface

Das Interface

Beim Erstellen der Interface-Datei habe ich mich bewusst auf die sehr ausgereifteISAPI - Schnittstelle des IIS konzentriert, denn man kann schon sagen, dass derIIS den größten Funktionsumfang zur Verfügung stellt in vielen anderen Implementierungen wie z.B. beim Apache wurde auf einige Möglichkeiten verzichtet, denn hier steht auch eine andere Schnittstelle zur Verfügung.

Wie schon erwähnt konnte ich es mir zunutze machen, dass bereits in C Header-Dateien zu diesem Interface zur Verfügung stehen, die nur nach Pascal portiert werden mussten. Im Internet findet man dazu einiges (wenn man nach isapi.pas sucht), aber leider sind die Ergebnisse nicht aktuell und auch leider nicht immer fehlerfrei und auch leider nicht direkt für Free Pascal zu gebrauchen.

Die aus den Recherchen im Internet entstandene Interface-Datei ist gleichermaßen für Lazarus / Free Pascal als auch für Delphi geeignet. Sie stellt die Definitionen der drei zu jeder Schnittstelle zu exportierten Funktionen zur Verfügung.

Bei dem Filter wären das:

1. GetFilterVersion, welche Informationen zum Filter übergibt, vor allem für welche Events das Filter geschrieben wurde.
2. HttpFilterProc die Hauptfunktion, welche eigentliche Verarbeitung vornimmt.
3. TerminateFilter, die das Beenden des Filters einleitet.

Bei der Extension wären das:

1. GetExtensionVersion, welche Informationen zur Extension übergibt.
2. HttpExtensionProc die Hauptfunktion, welche eigentliche Verarbeitung des Requests vornimmt.
3. TerminateExtension, die das Beenden der Extension einleitet.

isapi.pas Pascal (52,38 kByte) 04.09.2015 00:24

```
//
*************************************************************************
//  Title.............. :  Internet Server Application Programming Interface
v 7.0
//                         HTTP Server Extension interface v 7.0 (HttpExt.h)
//                         HTTP Server Filter interface v 7.0 (HttpFilt.h)
//  Modulname ......... :  isapi.pas
//  Type .............. :  Unit (Interface Library)
//  Author ............ :  Udo Schmal
//  Development Status  :  04.09.2015
//  Operating System .. :  Win32/64
//  IDE ............... :  Delphi & Lazarus
//
*************************************************************************

unit isapi;
{$WEAKPACKAGEUNIT}
{$ifdef fpc}
  {$mode objfpc}
{$endif}
{$H+}
```

```
interface

{$HPPEMIT '#include <httpext.h>'}
{$HPPEMIT '#include <httpfilt.h>'}

//uses Windows;
const
  MAX_PATH = 260;
type
  BOOL = longbool;
  LONG  = longint;
  LONGLONG  = int64;
  PWCHAR = Pwidechar;
  LPDWORD = ^DWORD;

const
  HSE_VERSION_MAJOR         =   7;    // major version of this spec
  {$EXTERNALSYM HSE_VERSION_MAJOR}
  HSE_VERSION_MINOR         =   0;    // minor version of this spec
  {$EXTERNALSYM HSE_VERSION_MINOR}
  HSE_LOG_BUFFER_LEN        =  80;
  {$EXTERNALSYM HSE_LOG_BUFFER_LEN}
  HSE_MAX_EXT_DLL_NAME_LEN  = 256;
  {$EXTERNALSYM HSE_MAX_EXT_DLL_NAME_LEN}

//  HSE_VERSION = MakeLong(HSE_VERSION_MINOR, HSE_VERSION_MAJOR);
  HSE_VERSION = $00070000;
  {$EXTERNALSYM HSE_VERSION}

// the following are the status codes returned by the Extension DLL
  HSE_STATUS_SUCCESS                = 1;
  {$EXTERNALSYM HSE_STATUS_SUCCESS}
  HSE_STATUS_SUCCESS_AND_KEEP_CONN = 2;
  {$EXTERNALSYM HSE_STATUS_SUCCESS_AND_KEEP_CONN}
  HSE_STATUS_PENDING                = 3;
  {$EXTERNALSYM HSE_STATUS_PENDING}
  HSE_STATUS_ERROR                  = 4;
  {$EXTERNALSYM HSE_STATUS_ERROR}

// The following are the values to request services with the
ServerSupportFunction.
// Values from 0 to 1000 are reserved for future versions of the interface
  HSE_REQ_BASE                              = 0;
  {$EXTERNALSYM HSE_REQ_BASE}
  HSE_REQ_SEND_URL_REDIRECT_RESP            = ( HSE_REQ_BASE + 1 );
  {$EXTERNALSYM HSE_REQ_SEND_URL_REDIRECT_RESP}
  HSE_REQ_SEND_URL                          = ( HSE_REQ_BASE + 2 );
  {$EXTERNALSYM HSE_REQ_SEND_URL}
  HSE_REQ_SEND_RESPONSE_HEADER              = ( HSE_REQ_BASE + 3 );
  {$EXTERNALSYM HSE_REQ_SEND_RESPONSE_HEADER}
  HSE_REQ_DONE_WITH_SESSION                 = ( HSE_REQ_BASE + 4 );
```

```
  {$EXTERNALSYM HSE_REQ_DONE_WITH_SESSION}
  HSE_REQ_END_RESERVED                  = 1000;
  {$EXTERNALSYM HSE_REQ_END_RESERVED}

// These are Microsoft specific extensions
  HSE_REQ_MAP_URL_TO_PATH               = (HSE_REQ_END_RESERVED+1);
  {$EXTERNALSYM HSE_REQ_MAP_URL_TO_PATH}
  HSE_REQ_GET_SSPI_INFO                 = (HSE_REQ_END_RESERVED+2);
  {$EXTERNALSYM HSE_REQ_GET_SSPI_INFO}
  HSE_APPEND_LOG_PARAMETER              = (HSE_REQ_END_RESERVED+3);
  {$EXTERNALSYM HSE_APPEND_LOG_PARAMETER}
  HSE_REQ_SEND_URL_EX                   = (HSE_REQ_END_RESERVED+4);
  {$EXTERNALSYM HSE_REQ_SEND_URL_EX}
  HSE_REQ_IO_COMPLETION                 = (HSE_REQ_END_RESERVED+5);
  {$EXTERNALSYM HSE_REQ_IO_COMPLETION}
  HSE_REQ_TRANSMIT_FILE                 = (HSE_REQ_END_RESERVED+6);
  {$EXTERNALSYM HSE_REQ_TRANSMIT_FILE}
  HSE_REQ_REFRESH_ISAPI_ACL             = (HSE_REQ_END_RESERVED+7);
  {$EXTERNALSYM HSE_REQ_REFRESH_ISAPI_ACL}
  HSE_REQ_IS_KEEP_CONN                  = (HSE_REQ_END_RESERVED+8);
  {$EXTERNALSYM HSE_REQ_IS_KEEP_CONN}
  HSE_REQ_ASYNC_READ_CLIENT             = (HSE_REQ_END_RESERVED+10);
  {$EXTERNALSYM HSE_REQ_ASYNC_READ_CLIENT}
  HSE_REQ_GET_IMPERSONATION_TOKEN       = (HSE_REQ_END_RESERVED+11);
  {$EXTERNALSYM HSE_REQ_GET_IMPERSONATION_TOKEN}
  HSE_REQ_MAP_URL_TO_PATH_EX            = (HSE_REQ_END_RESERVED+12);
  {$EXTERNALSYM HSE_REQ_MAP_URL_TO_PATH_EX}
  HSE_REQ_ABORTIVE_CLOSE                = (HSE_REQ_END_RESERVED+14);
  {$EXTERNALSYM HSE_REQ_ABORTIVE_CLOSE}
  HSE_REQ_GET_CERT_INFO_EX              = (HSE_REQ_END_RESERVED+15);
  {$EXTERNALSYM HSE_REQ_GET_CERT_INFO_EX}
  HSE_REQ_SEND_RESPONSE_HEADER_EX       = (HSE_REQ_END_RESERVED+16);
  {$EXTERNALSYM HSE_REQ_SEND_RESPONSE_HEADER_EX}
  HSE_REQ_CLOSE_CONNECTION              = (HSE_REQ_END_RESERVED+17);
  {$EXTERNALSYM HSE_REQ_CLOSE_CONNECTION}
  HSE_REQ_IS_CONNECTED                  = (HSE_REQ_END_RESERVED+18);
  {$EXTERNALSYM HSE_REQ_IS_CONNECTED}
  HSE_REQ_MAP_UNICODE_URL_TO_PATH       = (HSE_REQ_END_RESERVED+23);
  {$EXTERNALSYM HSE_REQ_MAP_UNICODE_URL_TO_PATH}
  HSE_REQ_MAP_UNICODE_URL_TO_PATH_EX    = (HSE_REQ_END_RESERVED+24);
  {$EXTERNALSYM HSE_REQ_MAP_UNICODE_URL_TO_PATH_EX}
  HSE_REQ_EXEC_UNICODE_URL              = (HSE_REQ_END_RESERVED+25);
  {$EXTERNALSYM HSE_REQ_EXEC_UNICODE_URL}
  HSE_REQ_EXEC_URL                      = (HSE_REQ_END_RESERVED+26);
  {$EXTERNALSYM HSE_REQ_EXEC_URL}
  HSE_REQ_GET_EXEC_URL_STATUS           = (HSE_REQ_END_RESERVED+27);
  {$EXTERNALSYM HSE_REQ_GET_EXEC_URL_STATUS}
  HSE_REQ_SEND_CUSTOM_ERROR             = (HSE_REQ_END_RESERVED+28);
  {$EXTERNALSYM HSE_REQ_SEND_CUSTOM_ERROR}
  HSE_REQ_IS_IN_PROCESS                 = (HSE_REQ_END_RESERVED+30);
  {$EXTERNALSYM HSE_REQ_IS_IN_PROCESS}
```

```
  HSE_REQ_REPORT_UNHEALTHY                  = (HSE_REQ_END_RESERVED+32);
  {$EXTERNALSYM HSE_REQ_REPORT_UNHEALTHY}
  HSE_REQ_NORMALIZE_URL                     = (HSE_REQ_END_RESERVED+33);
  {$EXTERNALSYM HSE_REQ_NORMALIZE_URL}
  HSE_REQ_VECTOR_SEND                       = (HSE_REQ_END_RESERVED+37);
  {$EXTERNALSYM HSE_REQ_VECTOR_SEND}
  HSE_REQ_GET_ANONYMOUS_TOKEN               = (HSE_REQ_END_RESERVED+38);
  {$EXTERNALSYM HSE_REQ_GET_ANONYMOUS_TOKEN}
  HSE_REQ_GET_CACHE_INVALIDATION_CALLBACK   = (HSE_REQ_END_RESERVED+40);
  {$EXTERNALSYM HSE_REQ_GET_CACHE_INVALIDATION_CALLBACK}
  HSE_REQ_GET_UNICODE_ANONYMOUS_TOKEN       = (HSE_REQ_END_RESERVED+41);
  {$EXTERNALSYM HSE_REQ_GET_UNICODE_ANONYMOUS_TOKEN}
  HSE_REQ_GET_TRACE_INFO                    = (HSE_REQ_END_RESERVED+42);
  {$EXTERNALSYM HSE_REQ_GET_TRACE_INFO}
  HSE_REQ_SET_FLUSH_FLAG                    = (HSE_REQ_END_RESERVED+43);
  {$EXTERNALSYM HSE_REQ_SET_FLUSH_FLAG}
  HSE_REQ_GET_TRACE_INFO_EX                 = (HSE_REQ_END_RESERVED+44);
  {$EXTERNALSYM HSE_REQ_RAISE_TRACE_EVENT}
  HSE_REQ_RAISE_TRACE_EVENT                 = (HSE_REQ_END_RESERVED+45);
  {$EXTERNALSYM HSE_REQ_GET_CONFIG_OBJECT}
  HSE_REQ_GET_CONFIG_OBJECT                 = (HSE_REQ_END_RESERVED+46);
  {$EXTERNALSYM HSE_REQ_GET_WORKER_PROCESS_SETTINGS}
  HSE_REQ_GET_WORKER_PROCESS_SETTINGS       = (HSE_REQ_END_RESERVED+47);
  {$EXTERNALSYM HSE_REQ_GET_PROTOCOL_MANAGER_CUSTOM_INTERFACE_CALLBACK}
  HSE_REQ_GET_PROTOCOL_MANAGER_CUSTOM_INTERFACE_CALLBACK =
(HSE_REQ_END_RESERVED+48);
  {$EXTERNALSYM HSE_REQ_CANCEL_IO}
  HSE_REQ_CANCEL_IO                         = (HSE_REQ_END_RESERVED+49);
  {$EXTERNALSYM HSE_REQ_GET_CHANNEL_BINDING_TOKEN}
  HSE_REQ_GET_CHANNEL_BINDING_TOKEN         = (HSE_REQ_END_RESERVED+50);

//  Bit Flags for TerminateExtension
//
//    HSE_TERM_ADVISORY_UNLOAD - Server wants to unload the extension,
//          extension can return TRUE if OK, FALSE if the server should not
//          unload the extension
//
//    HSE_TERM_MUST_UNLOAD - Server indicating the extension is about to be
//          unloaded, the extension cannot refuse.
  HSE_TERM_ADVISORY_UNLOAD                  = $00000001;
  {$EXTERNALSYM HSE_TERM_ADVISORY_UNLOAD}
  HSE_TERM_MUST_UNLOAD                      = $00000002;
  {$EXTERNALSYM HSE_TERM_MUST_UNLOAD}

// Flags for IO Functions, supported for IO Funcs.
//  TF means ServerSupportFunction( HSE_REQ_TRANSMIT_FILE)
  HSE_IO_SYNC                       = $00000001; // for WriteClient
  {$EXTERNALSYM HSE_IO_SYNC}
  HSE_IO_ASYNC                      = $00000002; // for WriteClient/TF
  {$EXTERNALSYM HSE_IO_ASYNC}
  HSE_IO_DISCONNECT_AFTER_SEND      = $00000004; // for TF
```

```
  {$EXTERNALSYM HSE_IO_DISCONNECT_AFTER_SEND}
  HSE_IO_SEND_HEADERS               = $00000008; // for TF
  {$EXTERNALSYM HSE_IO_SEND_HEADERS}
  HSE_IO_NODELAY                    = $00001000; // turn off TCP nagling
  {$EXTERNALSYM HSE_IO_NODELAY}
// These three are only used by VectorSend
  HSE_IO_FINAL_SEND                 = $00000010;
  {$EXTERNALSYM HSE_IO_FINAL_SEND}
  HSE_IO_CACHE_RESPONSE             = $00000020;
  {$EXTERNALSYM HSE_IO_CACHE_RESPONSE}
  HSE_IO_TRY_SKIP_CUSTOM_ERRORS     = $00000040;
  {$EXTERNALSYM HSE_IO_TRY_SKIP_CUSTOM_ERRORS}

type
  HCONN = THandle;
  {$EXTERNALSYM HCONN}

  // This structure passes IIS information regarding the version and
description
  // of the extension.
  PHSE_VERSION_INFO = ^HSE_VERSION_INFO;
  HSE_VERSION_INFO = packed record
    dwExtensionVersion: DWORD;
    lpszExtensionDesc: array [0..HSE_MAX_EXT_DLL_NAME_LEN-1] of Char;
  end;
  {$EXTERNALSYM HSE_VERSION_INFO}
  THSE_VERSION_INFO = HSE_VERSION_INFO;
  LPHSE_VERSION_INFO = PHSE_VERSION_INFO;
  {$EXTERNALSYM LPHSE_VERSION_INFO}

  // The GetServerVariable function retrieves information about an HTTP
  // connection or about IIS itself.
  // Some server variables, such as Request_Method and Content_Length are
  // embedded in the EXTENSION_CONTROL_BLOCK structure. You can use
  // GetServerVariable to obtain information about the request or server that
is
  // not included in EXTENSION_CONTROL_BLOCK.
  TGetServerVariableProc = function ( hConn: HCONN; VariableName: PChar;
    Buffer: Pointer; var Size: DWORD ): BOOL stdcall;

  // The WriteClient function is a callback function that is supplied in the
  // EXTENSION_CONTROL_BLOCK Structure for a request sent to the ISAPI
extension.
  // It sends the data present in the given buffer to the client that made
the request.
  TWriteClientProc = function ( ConnID: HCONN; Buffer: Pointer;
    var Bytes: DWORD; dwReserved: DWORD ): BOOL stdcall;

  // The ReadClient function reads data from the body of the client's HTTP
request.
  TReadClientProc = function ( ConnID: HCONN; Buffer: Pointer;
```

```
    var Size: DWORD ): BOOL stdcall;

  // The ServerSupportFunction function is a callback function that is
supplied
  // in the EXTENSION_CONTROL_BLOCK Structure that is associated with the
current
  // HTTP request. ServerSupportFunction can be used to perform a variety of
tasks.
  //  If a parameter is designated as unused in a particular support function,
  //  you should set that parameter to NULL.
  TServerSupportFunctionProc = function ( hConn: HCONN; HSERRequest: DWORD;
    Buffer: Pointer; Size: LPDWORD; DataType: LPDWORD ): BOOL stdcall;

  // This structure is used by IIS and the ISAPI extension to exchange
information.
  PEXTENSION_CONTROL_BLOCK = ^TEXTENSION_CONTROL_BLOCK;
  TEXTENSION_CONTROL_BLOCK = packed record
    cbSize: DWORD;                    // size of this struct.
    dwVersion: DWORD;                 // version info of this spec
    ConnID: HCONN;                    // Context number not to be modified!
    dwHttpStatusCode: DWORD;          // HTTP Status code
    lpszLogData: array [0..HSE_LOG_BUFFER_LEN-1] of Char; // null terminated
log info specific to this Extension DLL
{$ifdef WIN64}
    __padding: DWORD;
{$endif}
    lpszMethod: PChar;                // REQUEST_METHOD
    lpszQueryString: PChar;           // QUERY_STRING
    lpszPathInfo: PChar;              // PATH_INFO
    lpszPathTranslated: PChar;        // PATH_TRANSLATED
    cbTotalBytes: DWORD;              // Total bytes indicated from client
    cbAvailable: DWORD;               // Available number of bytes
    lpbData: Pointer;                 // pointer to cbAvailable bytes
    lpszContentType: PChar;           // Content type of client data

    GetServerVariable: TGetServerVariableProc;
    WriteClient: TWriteClientProc;
    ReadClient: TReadClientProc;
    ServerSupportFunction: TServerSupportFunctionProc;
  end;

//  Bit field of flags that can be on a virtual directory
const
  HSE_URL_FLAGS_READ          = $00000001; // Allow for Read
  {$EXTERNALSYM HSE_URL_FLAGS_READ}
  HSE_URL_FLAGS_WRITE         = $00000002; // Allow for Write
  {$EXTERNALSYM HSE_URL_FLAGS_WRITE}
  HSE_URL_FLAGS_EXECUTE       = $00000004; // Allow for Execute
  {$EXTERNALSYM HSE_URL_FLAGS_EXECUTE}
  HSE_URL_FLAGS_SSL           = $00000008; // Require SSL
  {$EXTERNALSYM HSE_URL_FLAGS_SSL}
```

```
  HSE_URL_FLAGS_DONT_CACHE     = $00000010; // Don't cache (vroot only)
  {$EXTERNALSYM HSE_URL_FLAGS_DONT_CACHE}
  HSE_URL_FLAGS_NEGO_CERT      = $00000020; // Allow client SSL certs
  {$EXTERNALSYM HSE_URL_FLAGS_NEGO_CERT}
  HSE_URL_FLAGS_REQUIRE_CERT   = $00000040; // Require client SSL certs
  {$EXTERNALSYM HSE_URL_FLAGS_REQUIRE_CERT}
  HSE_URL_FLAGS_MAP_CERT       = $00000080; // Map SSL cert to NT account
  {$EXTERNALSYM HSE_URL_FLAGS_MAP_CERT}
  HSE_URL_FLAGS_SSL128         = $00000100; // Require 128 bit SSL
  {$EXTERNALSYM HSE_URL_FLAGS_SSL128}
  HSE_URL_FLAGS_SCRIPT         = $00000200; // Allow for Script execution
  {$EXTERNALSYM HSE_URL_FLAGS_SCRIPT}
  HSE_URL_FLAGS_SCRIPT_SOURCE = $00000400; // Allow client to access script
source.
  {$EXTERNALSYM HSE_URL_FLAGS_SCRIPT_SOURCE}

  HSE_URL_FLAGS_MASK           = $000003FF;
  {$EXTERNALSYM HSE_URL_FLAGS_MASK}


type
  // You can use this structure with the HSE_REQ_MAP_URL_TO_PATH_EX function
of
  // the ServerSupportFunction. The structure returns information regarding a
  // virtual root that is to be mapped to a physical path.
  PHSE_URL_MAPEX_INFO = ^THSE_URL_MAPEX_INFO;
  THSE_URL_MAPEX_INFO = packed record
    lpszPath: array [0..MAX_PATH-1] of Char; // Physical path root mapped to
    dwFlags: DWORD;             // Flags associated with this URL path
    cchMatchingPath: DWORD;     // Number of matching characters in physical
path
    cchMatchingURL: DWORD;      // Number of matching characters in URL
    dwReserved1: DWORD;
    dwReserved2: DWORD;
  end;

  // You can use this structure with the HSE_REQ_MAP_UNICODE_URL_TO_PATH_EX
  // function of the ServerSupportFunction. The structure returns information
  // regarding a virtual root that is to be mapped to a physical path.
  PHSE_UNICODE_URL_MAPEX_INFO = ^THSE_UNICODE_URL_MAPEX_INFO;
  THSE_UNICODE_URL_MAPEX_INFO = packed record
    lpszPath: array [0..MAX_PATH-1] of WideChar; // Physical path root mapped
to
    dwFlags: DWORD;             // Flags associated with this URL path
    cchMatchingPath: DWORD;     // Number of matching characters in physical
path
    cchMatchingURL: DWORD;      // Number of matching characters in URL
  end;

  // Generally, if you use asynchronous I/O operations in your ISAPI
extension,
```

```
  // then you must provide a special asynchronous callback function of type
  // PFN_HSE_IO_COMPLETION. Unlike the other callback functions, this
function
  // must be provided and exposed by your extension, and will be called by
IIS.
  // When IIS completes an asynchronous read or write that your extension
  // requested, IIS calls your callback function. The function can be used to
  // read or write more data, or perform necessary resource de-allocation and
cleanup.
  THseIoCompletion = procedure (var ECB: TEXTENSION_CONTROL_BLOCK;
    pContext: Pointer; cbIO: DWORD; dwError: DWORD) stdcall;

  // This structure communicates information needed to use the Win32
  // TransmitFile function. It is supplied by the ISAPI extension when the
  // HSE_REQ_TRANSMIT_FILE parameter is specified for the dwHSERRequest
  // parameter of the ServerSupportFunction Function.
  PHSE_TF_INFO = ^THSE_TF_INFO;
  THSE_TF_INFO = record
    // callback and context information
    // the callback function will be called when IO is completed.
    // the context specified will be used during such callback.
    //
    // These values (if non-NULL) will override the one set by calling
    //  ServerSupportFunction() with HSE_REQ_IO_COMPLETION
    pfnHseIO: THseIoCompletion;
    pContext: Pointer;

    // file should have been opened with FILE_FLAG_SEQUENTIAL_SCAN
    hFile: THandle;

    // HTTP header and status code
    // These fields are used only if HSE_IO_SEND_HEADERS is present in dwFlags
    pszStatusCode: PChar; // HTTP Status Code  eg: "200 OK"

    BytesToWrite: DWORD;  // special value of "0" means write entire file.
    Offset: DWORD;        // offset value within the file to start from

    pHead: Pointer;       // Head buffer to be sent before file data
    HeadLength: DWORD;    // header length
{$ifdef WIN64}
    __padding: DWORD;     // don't know why
{$endif}
    pTail: Pointer;       // Tail buffer to be sent after file data
    TailLength: DWORD;    // tail length

    dwFlags: DWORD;       // includes HSE_IO_DISCONNECT_AFTER_SEND, ...
  end;

  // This structure is used by the ServerSupportFunction
  // HSE_REQ_SEND_RESPONSE_HEADER_EX. An ISAPI extension must fill the
  // structure with the required data and return it to IIS. You can use this
```

```
  // structure to indicate that the connection that is being used to service
the
  // current request should be kept active for further processing.
  PHSE_SEND_HEADER_EX_INFO = ^THSE_SEND_HEADER_EX_INFO;
  THSE_SEND_HEADER_EX_INFO = record

    // HTTP status code and header
    pszStatus: PChar;  // HTTP status code  eg: "200 OK"
    pszHeader: PChar;  // HTTP header

    cchStatus: DWORD;  // number of characters in status code
    cchHeader: DWORD;  // number of characters in header

    fKeepConn: BOOL;  // keep client connection alive?
{$ifdef WIN64}
    __padding: DWORD;
{$endif}
  end;

// Flags for use with HSE_REQ_EXEC_URL
const
  HSE_EXEC_URL_NO_HEADERS                   = $00000002;
  HSE_EXEC_URL_IGNORE_CURRENT_INTERCEPTOR   = $00000004;
  HSE_EXEC_URL_IGNORE_VALIDATION_AND_RANGE = $00000010;
  HSE_EXEC_URL_DISABLE_CUSTOM_ERROR         = $00000020;
  HSE_EXEC_URL_SSI_CMD                      = $00000040;
  HSE_EXEC_URL_HTTP_CACHE_ELIGIBLE          = $00000080;

type
  // This structure is used by a parent ISAPI extension to supply a user token
  // or user name to the child ISAPI extension.
  PHSE_EXEC_URL_USER_INFO = ^THSE_EXEC_URL_USER_INFO;
  THSE_EXEC_URL_USER_INFO = record
    hImpersonationToken: THandle;  //DWORD
    pszCustomUserName: PChar;
    pszCustomAuthType: PChar;
  end;

  // This structure is used by a parent ISAPI extension to choose and supply
an
  // entity body to the child ISAPI extension.
  // Because the child extension receives no information regarding the parent
  // ISAPI extension's actions, it is very important to send sufficient
  // information about the entity body to the child ISAPI extension.
  // For example, if the child has code to call ReadClient, send enough
  // information for the child to either call or not call ReadClient, based
on
  // knowledge that the client has, or does not have, further information to
send.
  PHSE_EXEC_URL_ENTITY_INFO = ^THSE_EXEC_URL_ENTITY_INFO;
  THSE_EXEC_URL_ENTITY_INFO = record
```

```
    cbAvailable: DWORD;
{$ifdef WIN64}
    __padding: DWORD;
{$endif}
    lpbData: Pointer;
  end;

  // The HSE_EXEC_URL_STATUS support structure contains information retrieved
by
  // HSE_REQ_GET_EXEC_URL_STATUS about a completed URL request.
  PHSE_EXEC_URL_STATUS = ^THSE_EXEC_URL_STATUS;
  THSE_EXEC_URL_STATUS = record
    uHttpStatusCode: WORD; //USHORT;
    uHttpSubStatus: WORD; //USHORT;
    dwWin32Error: DWORD;
  end;

  // The parent ISAPI extension uses this structure to specify how to invoke
a
  //child ISAPI extension when processing ANSI data.
  PHSE_EXEC_URL_INFO = ^THSE_EXEC_URL_INFO;
  THSE_EXEC_URL_INFO = record
    pszUrl: PChar;                      // URL to execute
    pszMethod: PChar;                   // Method
    pszChildHeaders: PChar;             // Request headers for child
    pUserInfo: THSE_EXEC_URL_USER_INFO; // User for new request
    pEntity: THSE_EXEC_URL_ENTITY_INFO; // Entity body for new request
    dwExecUrlFlags: DWORD;              // Flags
{$ifdef WIN64}
    __padding: DWORD;
{$endif}
  end;

  // When processing Unicode data, this structure is used by a parent ISAPI
  // extension to supply a user token to the child ISAPI extension.
  PHSE_EXEC_UNICODE_URL_USER_INFO = ^THSE_EXEC_UNICODE_URL_USER_INFO;
  THSE_EXEC_UNICODE_URL_USER_INFO = record
    hImpersonationToken: THandle;
    pszCustomUserName: PChar;
    pszCustomAuthType: PChar;
  end;

  // When processing Unicode data, this structure is used by a parent ISAPI
  // extension to specify how to invoke a child ISAPI extension.
  PHSE_EXEC_UNICODE_URL_INFO = ^THSE_EXEC_UNICODE_URL_INFO;
  THSE_EXEC_UNICODE_URL_INFO = record
    pszUrl: PWideChar;                  // URL to execute
    pszMethod: PChar;                   // Method
    pszChildHeaders: PChar;             // Request headers for child
    pUserInfo: THSE_EXEC_UNICODE_URL_USER_INFO; // User for new request
    pEntity: THSE_EXEC_URL_ENTITY_INFO; // Entity body for new request
```

```
    dwExecUrlFlags: DWORD;              // Flags
{$ifdef WIN64}
    __padding: DWORD;
{$endif}
  end;

  // The HSE_CUSTOM_ERROR_INFO support structure contains information used by
  // HSE_REQ_SEND_CUSTOM_ERROR.
  PHSE_CUSTOM_ERROR_INFO = ^THSE_CUSTOM_ERROR_INFO;
  THSE_CUSTOM_ERROR_INFO = record
    pszStatus: Char;
    uHttpSubError: WORD; //USHORT
{$ifdef WIN64}
    __padding: WORD;
{$endif}
    fAsync: BOOL; //WINBOOL
  end;

// structures for the HSE_REQ_VECTOR_SEND ServerSupportFunction

// Types of vector-elements currently supported
const
  HSE_VECTOR_ELEMENT_TYPE_MEMORY_BUFFER   = 0;
  HSE_VECTOR_ELEMENT_TYPE_FILE_HANDLE     = 1;
type
  // The HSE_VECTOR_ELEMENT structure contains data about elements of a
response
  // using HSE_REQ_VECTOR_SEND.
  PHSE_VECTOR_ELEMENT = ^THSE_VECTOR_ELEMENT;
  THSE_VECTOR_ELEMENT = record
    ElementType: DWORD; // type of element (buffer/file/fragment etc)
{$ifdef WIN64}
    __padding: DWORD;
{$endif}
    pvContext: Pointer; // the context representing the element to be sent
    cbOffset: LONGLONG; // offset from the start of hFile
    cbSize: LONGLONG;   // number of bytes to send
  end;

  PHSE_VECTOR_ELEMENT_Array = ^THSE_VECTOR_ELEMENT_Array;
  THSE_VECTOR_ELEMENT_Array = array[0..0] of THSE_VECTOR_ELEMENT;

  // The HSE_RESPONSE_VECTOR structure holds data about response elements
used
  // by the HSE_REQ_VECTOR_SEND support function.
  // If the HSE_IO_SEND_HEADERS flag is used, pszStatus and pszHeaders must
not
  // be NULL. If HSE_IO_SEND_HEADERS is not present, pszStatus and pszHeader
  // must be NULL. Any other combination is invalid.
  PHSE_RESPONSE_VECTOR = ^THSE_RESPONSE_VECTOR;
  THSE_RESPONSE_VECTOR = record
```

```
    dwFlags: DWORD;      // combination of HSE_IO_* flags
{$ifdef WIN64}
    __padding1: DWORD;
{$endif}
    pszStatus: PChar;    // status line to send like "200 OK"
    pszHeaders: PChar;   // headers to send
    nElementCount: DWORD;// number of THSE_VECTOR_ELEMENTs
{$ifdef WIN64}
    __padding2: DWORD;
{$endif}
    lpElementArray: THSE_VECTOR_ELEMENT_Array;// pointer to those elements
  end;

//typedef HRESULT (WINAPI *PFN_HSE_CACHE_INVALIDATION_CALLBACK)(WCHAR
*pszUrl);
  TInvalidationCallBack = function (pszUrl: WideChar): hresult stdcall;

(*** not implemented ***

#if(_WIN32_WINNT >= 0x400)
#include <wincrypt.h>

type
  // This structure is used to send certificate information to the ISAPI
  // extension when the HSE_REQ_GET_CERT_INFO_EX value is set for the
  // dwHSERRequest parameter of the ServerSupportFunction.
  PCERT_CONTEXT_EX = ^TCERT_CONTEXT_EX;
  TCERT_CONTEXT_EX = record
    CertContext: CERT_CONTEXT;
    cbAllocated: DWORD;
    dwCertificateFlags: DWORD;
  end;

**********************)

// HSE_TRACE_INFO structure used to get debug trace info
// from core web server
  PHSE_TRACE_INFO = ^THSE_TRACE_INFO;
  THSE_TRACE_INFO = record
    fTraceRequest: BOOL;  // Recommendation from IIS to trace the request
    TraceContextId: array[0..15] of Byte; // The unique trace context ID for
the current request
{$ifdef WIN64}
    __padding: DWORD;
{$endif}
    dwReserved1: DWORD;
    dwReserved2: DWORD;
  end;

// HSE_REQ_GET_TRACE_INFO_EX SSF uses
// the HTTP_TRACE_CONFIGURATION structure defined in httptrace.h
```

```
// HSE_REQ_RAISE_TRACE_EVENT SSF uses
// the HTTP_TRACE_EVENT structure defined in httptrace.h

// SSF_REQ_GET_WORKER_PROCESS_SETTINGS returns IWpfSettings pointer.
// IWpfSettings is defined in the wpframework.h

// SSF_REQ_GET_CONFIG_OBJECT returns INativeConfigurationSystem pointer.
// INativeConfigurationSystem is defined in the nativerd.h

// HSE_GET_PROTOCOL_MANAGER_CUSTOM_INTERFACE_CALLBACK returns pointer to
// PFN_HSE_GET_PROTOCOL_MANAGER_CUSTOM_INTERFACE_CALLBACK function type
  TProtocolManagerCustomInterfaceCallback = function (
    pszProtocolManagerDll: PWChar; pszProtocolManagerDllInitFunction: PWChar;
    dwCustomInterfaceId: DWORD; ppCustomInterface: Pointer): hresult stdcall;

const
// Flags for determining application type
  HSE_APP_FLAG_IN_PROCESS   = 0;
  HSE_APP_FLAG_ISOLATED_OOP = 1;
  HSE_APP_FLAG_POOLED_OOP   = 2;

type
//  these are the prototypes that must be exported from the extension DLL

//  function GetExtensionVersion(var Ver: THSE_VERSION_INFO): BOOL; stdcall;
//  function HttpExtensionProc(var ECB: TEXTENSION_CONTROL_BLOCK): DWORD;
stdcall;
//  function TerminateExtension(dwFlags: DWORD): BOOL; stdcall;

// the following type declarations are for the server side

  // The GetExtensionVersion function is the first entry-point function in
IIS.
  // This function allows your ISAPI extension to register its version
  // information with IIS.
  TGetExtensionVersion = function (var Ver: THSE_VERSION_INFO): BOOL stdcall;

  // The HttpExtensionProc function is the main entry point for an ISAPI
  // extension called by IIS. It exposes methods that IIS uses to access the
  // functionality exposed by the extension.
  THttpExtensionProc = function (var ECB: TEXTENSION_CONTROL_BLOCK): DWORD
stdcall;

  // The TerminateExtension function unloads the ISAPI DLL. This optional
  // entry-point function is called by IIS immediately before unloading the
DLL
  // from its process.
  TTerminateExtension = function (dwFlags: DWORD): BOOL stdcall;

  // Interface structure for an isapi extension
```

```
  IISAPIExtension = interface
    function GetExtensionVersion(var Ver: THSE_VERSION_INFO): BOOL; stdcall;
    function HttpExtensionProc(var ECB: TEXTENSION_CONTROL_BLOCK): DWORD;
stdcall;
    function TerminateExtension(dwFlags: DWORD): BOOL; stdcall;
  end;

//_____________________________________________________________________

// from here module contains the Microsoft HTTP filter extension info

//  Current version of the filter spec is 7.0
const
  HTTP_FILTER_REVISION = $00070000;
  {$EXTERNALSYM HTTP_FILTER_REVISION}

  SF_MAX_USERNAME         = (256+1);
  {$EXTERNALSYM SF_MAX_USERNAME}
  SF_MAX_PASSWORD         = (256+1);
  {$EXTERNALSYM SF_MAX_PASSWORD}
  SF_MAX_AUTH_TYPE = (32+1);
  {$EXTERNALSYM SF_MAX_AUTH_TYPE}

  SF_MAX_FILTER_DESC_LEN  = (256+1);
  {$EXTERNALSYM SF_MAX_FILTER_DESC_LEN}

  //  These values can be used with the pfnSFCallback function supplied in
  //  the filter context structure

//  SF_REQ_TYPE

  //  Sends a complete HTTP server response header including
  //  the status, server version, message time and MIME version.
  //
  //  Server extensions should append other information at the end,
  //  such as Content-type, Content-length etc followed by an extra
  //  '\r\n'.
  //
  //  pData - Zero terminated string pointing to optional
  //      status string (i.e., "401 Access Denied") or NULL for
  //      the default response of "200 OK".
  //
  //  ul1 - Zero terminated string pointing to optional data to be
  //      appended and set with the header.  If NULL, the header will
  //      be terminated with an empty line.
  SF_REQ_SEND_RESPONSE_HEADER = 0;
  {$EXTERNALSYM SF_REQ_SEND_RESPONSE_HEADER}

  //  If the server denies the HTTP request, add the specified headers
  //  to the server error response.
  //
```

```
//  This allows an authentication filter to advertise its services
//  w/o filtering every request.  Generally the headers will be
//  WWW-Authenticate headers with custom authentication schemes but
//  no restriction is placed on what headers may be specified.
//
//  pData - Zero terminated string pointing to one or more header lines
//      with terminating '\r\n'.
SF_REQ_ADD_HEADERS_ON_DENIAL = 1;
{$EXTERNALSYM SF_REQ_ADD_HEADERS_ON_DENIAL}

//  Only used by raw data filters that return SF_STATUS_READ_NEXT
//  ul1 - size in bytes for the next read
SF_REQ_SET_NEXT_READ_SIZE = 2;
{$EXTERNALSYM SF_REQ_SET_NEXT_READ_SIZE}

//  Used to indicate this request is a proxy request
//  ul1 - The proxy flags to set
//      0x00000001 - This is a HTTP proxy request
SF_REQ_SET_PROXY_INFO = 3;
{$EXTERNALSYM SF_REQ_SET_PROXY_INFO}

//  Returns the connection ID contained in the ConnID field of an
//  ISAPI Application's Extension Control Block.  This value can be used
//  as a key to cooridinate shared data between Filters and Applications.
//  pData - Pointer to DWORD that receives the connection ID.
SF_REQ_GET_CONNID = 4;
{$EXTERNALSYM SF_REQ_GET_CONNID}

// Used to set a SSPI security context + impersonation token
// derived from a client certificate.
// pData - certificate info ( PHTTP_FILTER_CERTIFICATE_INFO )
// ul1 - CtxtHandle*
// ul2 - impersonation handle
SF_REQ_SET_CERTIFICATE_INFO = 5;
{$EXTERNALSYM SF_REQ_SET_CERTIFICATE_INFO}

// Used to get an IIS property as defined in SF_PROPERTY_IIS
// ul1 - Property ID
SF_REQ_GET_PROPERTY = 6;
{$EXTERNALSYM SF_REQ_GET_PROPERTY}

// Used to normalize an URL
// pData - URL to normalize
SF_REQ_NORMALIZE_URL = 7;
{$EXTERNALSYM SF_REQ_NORMALIZE_URL}

// Disable Notifications
// ul1 - notifications to disable
SF_REQ_DISABLE_NOTIFICATIONS = 8;
{$EXTERNALSYM SF_REQ_DISABLE_NOTIFICATIONS}
```

```
// SF_PROPERTY_IIS
  SF_PROPERTY_SSL_CTXT = 0;
  {$EXTERNALSYM SF_PROPERTY_SSL_CTXT}
  SF_PROPERTY_INSTANCE_NUM_ID = 1;
  {$EXTERNALSYM SF_PROPERTY_INSTANCE_NUM_ID}

  //  These values are returned by the filter entry point when a new request
is
  //  received indicating their interest in this particular request

// SF_STATUS_TYPE
  // The filter has handled the HTTP request. The server should disconnect
the
  // session.
  SF_STATUS_REQ_FINISHED = $8000000;
  {$EXTERNALSYM SF_STATUS_REQ_FINISHED}

  // This filter is not supported. Same as SF_STATUS_FINISHED except the
server
  // should keep the TCP session open if the option was negotiated
  SF_STATUS_REQ_FINISHED_KEEP_CONN = $8000001;
  {$EXTERNALSYM SF_STATUS_REQ_FINISHED_KEEP_CONN}

  // The next filter in the notification chain should be called.
  SF_STATUS_REQ_NEXT_NOTIFICATION = $8000002;
  {$EXTERNALSYM SF_STATUS_REQ_NEXT_NOTIFICATION}

  // This filter handled the notification. No other handlers should be called
  // for this particular notification type.
  SF_STATUS_REQ_HANDLED_NOTIFICATION = $8000003;
  {$EXTERNALSYM SF_STATUS_REQ_HANDLED_NOTIFICATION}

  // Tells IIS that an error occurred during the processing of the request.
  // The filter should call SetLastError with the nature of the failure,
  // otherwise the client might get an unexpected response. IIS looks at
  // GetLastError and the notification where the error occurred to decide
what
  // error to send to the client. For example, if the filter calls
SetLastError
  // with a "File not found" error, IIS will return a 404 to the client.
  SF_STATUS_REQ_ERROR = $8000004;
  {$EXTERNALSYM SF_STATUS_REQ_ERROR}

  // The filter is an opaque stream filter (encrypted/compressed HTTP
requests)
  // and the session parameters are being negotiated. This is valid only for
  // raw-read notification. This notification indicates that the full request
  // has not yet been received; the Web server should issue another read and
  // notify the filter with the additional data read.
  SF_STATUS_REQ_READ_NEXT = $8000005;
  {$EXTERNALSYM SF_STATUS_REQ_READ_NEXT}
```

```
// pvNotification points to this structure for all request notification types
type

  // The GetServerVariable callback function specifies a server variable that
  // the ISAPI filter needs to retrieve from IIS.
//  TGetServerVariable = function (var pfc{: THTTP_FILTER_CONTEXT};
//    lpszName: PChar; var lpvBuffer; var lpdwSize: DWORD): BOOL stdcall;
  TFilterGetServerVariableProc = function (var pfc{: THTTP_FILTER_CONTEXT};
    VariableName: PChar; Buffer: Pointer; var Size: DWORD ): BOOL stdcall;

  // The AddResponseHeaders callback function specifies a response header for
  // IIS to send to the client.
  TFilterAddResponseHeadersProc = function (var pfc{: THTTP_FILTER_CONTEXT};
    lpszHeaders: PChar; dwReserved: DWORD): BOOL stdcall;

  // The WriteClient callback function is called by an ISAPI filter to send
data
  // to the client.
  TFilterWriteClientProc = function (var pfc{: THTTP_FILTER_CONTEXT};
    Buffer: Pointer; var Bytes: DWORD; dwReserved: DWORD ): BOOL stdcall;

  // The AllocMem callback function allocates memory from the process heap to
a
  // buffer. Any memory allocated with this function will automatically be
freed
  // by IIS when the session ends.
  TFilterAllocMemProc = function (var pfc{: THTTP_FILTER_CONTEXT};
    cbSize: DWORD; dwReserved: DWORD): Pointer stdcall;

  // The ServerSupportFunction callback function can be used by ISAPI filters
to
  // accomplish a wide variety of tasks.
  // If a parameter is designated as unused for a particular support function,
  // you should set the parameter to NULL or 0, as appropriate.
  TFilterServerSupportFunctionProc = function (var pfc{:
THTTP_FILTER_CONTEXT};
    sfReq: DWORD; pData: Pointer; ul1, ul2: DWORD): BOOL stdcall;

  // This structure is used by HTTP_FILTER_PROC to obtain information about
the
  // current request. This structure is very similar in function to an ISAPI
  // extension's EXTENSION_CONTROL_BLOCK structure.
  PHTTP_FILTER_CONTEXT = ^THTTP_FILTER_CONTEXT;
  THTTP_FILTER_CONTEXT = record
    cbSize: DWORD;
    Revision: DWORD;          // This is the structure revision level.
    ServerContext: Pointer;   // Private context information for the server.
    ulReserved: DWORD;
    fIsSecurePort: BOOL;      // TRUE if this request is coming over a secure
port
```

```
    pFilterContext: Pointer; // A context that can be used by the filter
    // Server callbacks
    GetServerVariable: TFilterGetServerVariableProc;
    AddResponseHeaders: TFilterAddResponseHeadersProc;
    WriteClient: TFilterWriteClientProc;
    AllocMem: TFilterAllocMemProc;
    ServerSupportFunction: TFilterServerSupportFunctionProc;
  end;
  HTTP_FILTER_CONTEXT = THTTP_FILTER_CONTEXT;

  // IIS includes a pointer to this structure when it is either reading or
  // sending raw data. If your filter should be notified for this event, it
  // should register for either the SF_NOTIFY_READ_RAW_DATA or
  // SF_NOTIFY_SEND_RAW_DATA events.
  PHTTP_FILTER_RAW_DATA = ^HTTP_FILTER_RAW_DATA;
  HTTP_FILTER_RAW_DATA = record
    pvInData: Pointer;    // This is a pointer to the data for the filter to
process.
    cbInData: DWORD;      // Number of valid data bytes
    cbInBuffer: DWORD;    // Total size of buffer
    dwReserved: DWORD;
  end;
  {$EXTERNALSYM HTTP_FILTER_RAW_DATA}
  THTTP_FILTER_RAW_DATA = HTTP_FILTER_RAW_DATA;
  LPHTTP_FILTER_RAW_DATA = PHTTP_FILTER_RAW_DATA;
  {$EXTERNALSYM LPHTTP_FILTER_RAW_DATA}

  // The GetHeader callback function retrieves a header from IIS.
  //  Header names should include the trailing ':'. The special values
  //  'method', 'url' and 'version' can be used to retrieve the individual
  //  portions of the request line
  TGetHeaderProc = function (var pfc: THTTP_FILTER_CONTEXT; lpszName: PChar;
    out lpvBuffer; out lpdwSize: DWORD): BOOL stdcall;

  // The SetHeader callback function is used by ISAPI filters to change or
  // delete the value of a header. The function can be used to change the
  // special values included in the request line.
  //  To delete a header, specified a value of '\0'.
  TSetHeaderProc = function (var pfc: THTTP_FILTER_CONTEXT; lpszName,
    lpszValue: PChar): BOOL stdcall;

  // The AddHeader callback function adds an HTTP header to the incoming
request
  // or outgoing response.
  TAddHeaderProc = function (var pfc: THTTP_FILTER_CONTEXT; lpszName,
    lpszValue: PChar): BOOL; stdcall;

  // IIS includes a pointer to this structure when it is preprocessing a
  // request's headers. If your filter should be notified for this event, it
  // should register for the SF_NOTIFY_PREPROC_HEADERS event.
  PHTTP_FILTER_PREPROC_HEADERS = ^HTTP_FILTER_PREPROC_HEADERS;
```

```
  HTTP_FILTER_PREPROC_HEADERS = record
    GetHeader: TGetHeaderProc;
    SetHeader: TSetHeaderProc;
    AddHeader: TAddHeaderProc;
    HttpStatus: DWORD; // New in 4.0, status for SEND_RESPONSE
    dwReserved: DWORD; // New in 4.0
  end;
  {$EXTERNALSYM HTTP_FILTER_PREPROC_HEADERS}
  THTTP_FILTER_PREPROC_HEADERS = HTTP_FILTER_PREPROC_HEADERS;
  LPHTTP_FILTER_PREPROC_HEADERS = PHTTP_FILTER_PREPROC_HEADERS;
  {$EXTERNALSYM LPHTTP_FILTER_PREPROC_HEADERS}

  // ISAPI filters receive this notification immediately prior to sending the
  // headers to the client. The filter can inspect, modify, or add headers
the
  // client will receive as part of the response to the client's original
  // request. The structure contains the same members as
  // HTTP_FILTER_PREPROC_HEADERS. When a filter has registered for the
  // SF_NOTIFY_SEND_RESPONSE event, the pvNotification parameter of
  // HttpFilterProc will point to this structure.
  PHTTP_FILTER_SEND_RESPONSE = ^HTTP_FILTER_SEND_RESPONSE;
  HTTP_FILTER_SEND_RESPONSE = HTTP_FILTER_PREPROC_HEADERS;
  {$EXTERNALSYM HTTP_FILTER_SEND_RESPONSE}
  THTTP_FILTER_SEND_RESPONSE = HTTP_FILTER_SEND_RESPONSE;
  LPHTTP_FILTER_SEND_RESPONSE = PHTTP_FILTER_SEND_RESPONSE;
  {$EXTERNALSYM LPHTTP_FILTER_SEND_RESPONSE}

  // IIS includes a pointer to this structure when it is authenticating a
user
  // with either anonymous or Basic authentication schemes. If your filter
  // should be notified for this event, it should register for the
  // SF_NOTIFY_AUTHENTICATION event.
  PHTTP_FILTER_AUTHENT = ^HTTP_FILTER_AUTHENT;
  HTTP_FILTER_AUTHENT = record
    //  Pointer to username and password, empty strings for the anonymous user
    //  Client's can overwrite these buffers which are guaranteed to be at
    //  least SF_MAX_USERNAME and SF_MAX_PASSWORD bytes large.
    pszUser: PChar;
    cbUserBuff: DWORD;
    pszPassword: PChar;
    cbPasswordBuff: DWORD;
  end;
  {$EXTERNALSYM HTTP_FILTER_AUTHENT}
  THTTP_FILTER_AUTHENT = HTTP_FILTER_AUTHENT;
  LPHTTP_FILTER_AUTHENT = PHTTP_FILTER_AUTHENT;
  {$EXTERNALSYM LPHTTP_FILTER_AUTHENT}

  // IIS includes a pointer to this structure when it maps a URL to a
physical
  // directory. If your filter should be notified for this event, it should
  // register for the SF_NOTIFY_URL_MAP event.
```

```
  PHTTP_FILTER_URL_MAP = ^HTTP_FILTER_URL_MAP;
  HTTP_FILTER_URL_MAP = record
    pszURL: PChar; // const
    pszPhysicalPath: PChar;
    cbPathBuff: DWORD;
  end;
  {$EXTERNALSYM HTTP_FILTER_URL_MAP}
  THTTP_FILTER_URL_MAP = HTTP_FILTER_URL_MAP;
  LPHTTP_FILTER_URL_MAP = PHTTP_FILTER_URL_MAP;
  {$EXTERNALSYM LPHTTP_FILTER_URL_MAP}

  //  Indicates the server is going to use the specific physical mapping for
  //  the specified URL.  Filters can modify the physical path in place.
  //  Additional members beyond those from HTTP_FILTER_URL_MAP are
  //  informational.
  PHTTP_FILTER_URL_MAP_EX = ^HTTP_FILTER_URL_MAP_EX;
  HTTP_FILTER_URL_MAP_EX = record
    pszURL: PChar; // const
    pszPhysicalPath: PChar;
    cbPathBuff: DWORD;
    dwFlags: DWORD; // The AccessPerm metabase property that applies to this
URL
    cchMatchingPath: DWORD; // Number of matching characters in physical path
corresponding to the metabase node that applies.
    cchMatchingURL: DWORD; // Number of matching characters in the URL
corresponding to the metabase node that applies.
    pszScriptMapEntry: PChar; // const The physical path of the dll or exe
that to which this URL is script mapped.  This member will be NULL if no
script map applies.
  end;
  {$EXTERNALSYM HTTP_FILTER_URL_MAP_EX}
  THTTP_FILTER_URL_MAP_EX = HTTP_FILTER_URL_MAP_EX;
  LPHTTP_FILTER_URL_MAP_EX = PHTTP_FILTER_URL_MAP_EX;
  {$EXTERNALSYM LPHTTP_FILTER_URL_MAP_EX}

const
  // Bitfield indicating the requested resource has been denied by the server
due
  // to a logon failure, an ACL on a resource, an ISAPI Filter or an
  // ISAPI Application/CGI Application.
  //
  // SF_DENIED_BY_CONFIG can appear with SF_DENIED_LOGON if the server
  // configuration did not allow the user to logon.
  SF_DENIED_LOGON             = $00000001;
  {$EXTERNALSYM SF_DENIED_LOGON}
  SF_DENIED_RESOURCE          = $00000002;
  {$EXTERNALSYM SF_DENIED_RESOURCE}
  SF_DENIED_FILTER            = $00000004;
  {$EXTERNALSYM SF_DENIED_FILTER}
  SF_DENIED_APPLICATION       = $00000008;
  {$EXTERNALSYM SF_DENIED_APPLICATION}
```

```
  SF_DENIED_BY_CONFIG          = $00010000;
  {$EXTERNALSYM SF_DENIED_BY_CONFIG}

type
  // IIS includes a pointer to this structure when a user is presented with
an
  // Access Denied error message. If your filter should be notified for this
  // event, it should register for the SF_NOTIFY_ACCESS_DENIED event.
  PHTTP_FILTER_ACCESS_DENIED = ^HTTP_FILTER_ACCESS_DENIED;
  HTTP_FILTER_ACCESS_DENIED = record
    pszURL: PChar; // const Requesting URL
    pszPhysicalPath: PChar; // const Physical path of resource
    dwReason: DWORD; // Bitfield of SF_DENIED flags
  end;
  {$EXTERNALSYM HTTP_FILTER_ACCESS_DENIED}
  THTTP_FILTER_ACCESS_DENIED = HTTP_FILTER_ACCESS_DENIED;
  LPHTTP_FILTER_ACCESS_DENIED = PHTTP_FILTER_ACCESS_DENIED;
  {$EXTERNALSYM LPHTTP_FILTER_ACCESS_DENIED}

  // IIS includes a pointer to this structure when it is writing information
to
  // a log file. If your filter should be notified for this event, it should
  // register for the SF_NOTIFY_LOG event.
  PHTTP_FILTER_LOG = ^HTTP_FILTER_LOG;
  HTTP_FILTER_LOG = record
    pszClientHostName: PChar; // const clients's host name (c-ip)
    pszClientUserName: PChar; // const client's user name (c-username)
    pszServerName: PChar; // const name of the server to witch the client is
connected (c-computername)
    pszOperation: PChar; // const HTTP operation type (cs-method)
    pszTarget: PChar; // const target of the HTTP command (cs-uri-stem)
    pszParameters: PChar; // const parrameters passed to the HTTP command (cs-
uri-query)
    dwHttpStatus: DWORD; // HTTP return status (sc-status)
    dwWin32Status: DWORD; // Win32 error code (sc-win32-status)
    dwBytesSent: DWORD; // IIS 4.0 and later (sc-bytes)
    dwBytesRecvd: DWORD; // IIS 4.0 and later (cs-bytes)
    msTimeForProcessing: DWORD; // IIS 4.0 and later (time-taken)
  end;
  {$EXTERNALSYM HTTP_FILTER_LOG}
  THTTP_FILTER_LOG = HTTP_FILTER_LOG;
  LPHTTP_FILTER_LOG = PHTTP_FILTER_LOG;
  {$EXTERNALSYM LPHTTP_FILTER_LOG}

  //  Get the authenticated user impersonation token
  TGetUserTokenProc = function (var pfc: THTTP_FILTER_CONTEXT; phToken:
THandle): BOOL stdcall;

  // IIS includes a pointer to this structure when it is preprocessing a
  // request's headers. If your filter should be notified for this event,
  // it should register for the SF_NOTIFY_AUTH_COMPLETE event, which occurs
```

```
  // after the client's identity has been authenticated.
  //  Header names should include the trailing ':'.  The special values
  //  'method', 'url' and 'version' can be used to retrieve the individual
  //  portions of the request line
  PHTTP_FILTER_AUTH_COMPLETE_INFO = ^HTTP_FILTER_AUTH_COMPLETE_INFO;
  HTTP_FILTER_AUTH_COMPLETE_INFO = record
    GetHeader: TGetHeaderProc;
    SetHeader: TSetHeaderProc;
    AddHeader: TAddHeaderProc;
    GetUserToken: TGetUserTokenProc;
    HttpStatus: DWORD;
    fResetAuth: BOOL;
    dwReserved: DWORD;
  end;
  {$EXTERNALSYM HTTP_FILTER_AUTH_COMPLETE_INFO}
  THTTP_FILTER_AUTH_COMPLETE_INFO = HTTP_FILTER_AUTH_COMPLETE_INFO;
  LPHTTP_FILTER_AUTH_COMPLETE_INFO = PHTTP_FILTER_AUTH_COMPLETE_INFO;
  {$EXTERNALSYM LPHTTP_FILTER_AUTH_COMPLETE_INFO}

const
  //  Notification Flags

  // Indicates whether the application wants to be notified for transactions
  // that are happenning on the server port(s) that support data encryption
  // (such as PCT and SSL), on only the non-secure port(s) or both.

  // Notify the application only for connections over a secure port.
  SF_NOTIFY_SECURE_PORT               = $00000001;
  {$EXTERNALSYM SF_NOTIFY_SECURE_PORT}

  // Notify the application only for connections over a nonsecure port.
  SF_NOTIFY_NONSECURE_PORT            = $00000002;
  {$EXTERNALSYM SF_NOTIFY_NONSECURE_PORT}

  // When a client sends a request, one or more SF_NOTIFY_READ_RAW_DATA
  // notifications occur. Data is read until the client has sent all of the
  // HTTP headers associated with the request.
  SF_NOTIFY_READ_RAW_DATA             = $00008000;
  {$EXTERNALSYM SF_NOTIFY_READ_RAW_DATA}

  // A single SF_NOTIFY_PREPROC_HEADERS notification occurs for each request.
  // This notification indicates that the server has completed preprocessing
  // of the headers associated with the request, but has not yet begun to
  //process the information in the headers.
  SF_NOTIFY_PREPROC_HEADERS           = $00004000;
  {$EXTERNALSYM SF_NOTIFY_PREPROC_HEADERS}

  // An SF_NOTIFY_URL_MAP notification occurs whenever the server is
converting
  // a URL to a physical path. This notification occurs at least once after
the
```

```
  // preprocessed header's notification for the request, and might occur many
  // additional times during processing of the associated request.
  SF_NOTIFY_URL_MAP                   = $00001000;
  {$EXTERNALSYM SF_NOTIFY_URL_MAP}

  // An SF_NOTIFY_AUTHENTICATION notification occurs just before IIS attempts
to
  // authenticate the client. This notification occurs for every new
connection
  // (including anonymous requests), and every time the client sends enabled
  // user credentials for the target URL, in the form of an authorization
header,
  // to be authorized by the server. The AuthPersistence property setting in
the
  // metabase directly affects this filter. Note that not all requests are
  // guaranteed to trigger an authentication notification. This notification
  // only fires for anonymous requests and requests with an authorization
header
  // that specifies Basic authentication.
  SF_NOTIFY_AUTHENTICATION            = $00002000;
  {$EXTERNALSYM SF_NOTIFY_AUTHENTICATION}

  // If a 401 response is sent to the client, the SF_NOTIFY_ACCESS_DENIED
event
  // notification occurs. With this notification, the order of events changes.
  // The next event is usually SF_NOTIFY_END_OF_REQUEST.
  SF_NOTIFY_ACCESS_DENIED             = $00000800;
  {$EXTERNALSYM SF_NOTIFY_ACCESS_DENIED}

  // This notification, new to IIS 5.0, offers functionality similar to that
of
  // SF_NOTIFY_PREPROC_HEADERS. Specifically, it allows viewing and
modification
  // of the method, URL, version, or headers sent from the client. The key
  // difference between this notification and preprocessed headers is that
this
  // notification occurs after the client's identity has been negotiated with
  // the client. Because of the notification's timing, the AUTH_USER server
  // variable can be used to reliably obtain the identity of the user. Also,
  // functionality is provided to retrieve a copy of the token that IIS
  // impersonates when processing the request.
  SF_NOTIFY_AUTH_COMPLETE             = $04000000;
  {$EXTERNALSYM SF_NOTIFY_AUTH_COMPLETE}

  // This event occurs after the request is processed and before headers are
  // sent back to the client.
  SF_NOTIFY_SEND_RESPONSE             = $00000040;
  {$EXTERNALSYM SF_NOTIFY_SEND_RESPONSE}

  // As the request handler returns data to the client, one or more
  // SF_NOTIFY_SEND_RAW_DATA notifications occur.
```

```
  SF_NOTIFY_SEND_RAW_DATA             = $00000400;
  {$EXTERNALSYM SF_NOTIFY_SEND_RAW_DATA}

  // At the end of each request, the SF_NOTIFY_END_OF_REQUEST notification
occurs.
  SF_NOTIFY_END_OF_REQUEST            = $00000080;
  {$EXTERNALSYM SF_NOTIFY_END_OF_REQUEST}

  // After the HTTP request is complete and just before IIS writes the
request
  // to its log, the SF_NOTIFY_LOG notification occurs.
  SF_NOTIFY_LOG                       = $00000200;
  {$EXTERNALSYM SF_NOTIFY_LOG}

  // When the connection between the client and the server is closed, the
  // SF_NOTIFY_END_OF_NET_SESSION notification occurs. If a Keep-Alive
  // connection has been negotiated, it is possible for many HTTP requests to
  // occur before this notification.
  SF_NOTIFY_END_OF_NET_SESSION        = $00000100;
  {$EXTERNALSYM SF_NOTIFY_END_OF_NET_SESSION}


  //  Filter ordering flags
  //
  //  Filters will tend to be notified by their specified
  //  ordering.  For ties, notification order is determined by load order.
  //
  //  SF_NOTIFY_ORDER_HIGH - Authentication or data transformation filters
  //  SF_NOTIFY_ORDER_MEDIUM
  //  SF_NOTIFY_ORDER_LOW  - Logging filters that want the results of any
other
  //                         filters might specify this order.
  SF_NOTIFY_ORDER_HIGH                = $00080000;
  {$EXTERNALSYM SF_NOTIFY_ORDER_HIGH}
  SF_NOTIFY_ORDER_MEDIUM              = $00040000;
  {$EXTERNALSYM SF_NOTIFY_ORDER_MEDIUM}
  SF_NOTIFY_ORDER_LOW                 = $00020000;
  {$EXTERNALSYM SF_NOTIFY_ORDER_LOW}
  SF_NOTIFY_ORDER_DEFAULT             = SF_NOTIFY_ORDER_LOW;
  {$EXTERNALSYM SF_NOTIFY_ORDER_DEFAULT}
  SF_NOTIFY_ORDER_MASK                = SF_NOTIFY_ORDER_HIGH or
                                        SF_NOTIFY_ORDER_MEDIUM or
                                        SF_NOTIFY_ORDER_LOW;
  {$EXTERNALSYM SF_NOTIFY_ORDER_MASK}

type
  // This structure is used by GetFilterVersion to obtain the version
  // information about the filter.
  PHTTP_FILTER_VERSION = ^HTTP_FILTER_VERSION;
  HTTP_FILTER_VERSION = record
    dwServerFilterVersion: DWORD; // Version of the spec the server is using
```

```
    dwFilterVersion: DWORD;       // Fields specified by the client
    lpszFilterDesc: array[0..SF_MAX_FILTER_DESC_LEN - 1] of Char;
    dwFlags: DWORD;
  end;
  {$EXTERNALSYM HTTP_FILTER_VERSION}
  THTTP_FILTER_VERSION = HTTP_FILTER_VERSION;
  LPHTTP_FILTER_VERSION = PHTTP_FILTER_VERSION;
  {$EXTERNALSYM LPHTTP_FILTER_VERSION}

// A filter DLL's entry point looks like this.
// The return code should be an SF_STATUS_TYPE
//
// NotificationType - Type of notification
// pvNotification - Pointer to notification specific data
//
// function GetFilterVersion(var pVer: THTTP_FILTER_VERSION): BOOL; stdcall;
//
// function HttpFilterProc(var pfc: THTTP_FILTER_CONTEXT; Notificationtype:
DWORD;
//   pvNotification: Pointer): DWORD; stdcall;
//
// function TerminateFilter(dwFlags: DWORD): BOOL; stdcall;

// the following type declarations are for the server side

  // The GetFilterVersion function is the first entry-point function called
by
  // IIS on your ISAPI filter, and must be present for the filter to work
  // properly. IIS passes a pointer to a HTTP_FILTER_VERSION Structure, which
  // can be used to supply important filter configuration information to IIS.
  // The most important information passed to IIS is the bitmask that
contains
  // flags that specify which notification events your filter can process,
and
  // a flag that indicates the overall processing priority for your filter.
  TGetFilterVersion = function (var pVer: THTTP_FILTER_VERSION): BOOL stdcall;

  // IIS calls the HttpFilterProc entry-point function whenever a
notification
  // event for which the filter has registered occurs. IIS uses this function
to
  // pass information and control to your ISAPI filter.
  THttpFilterProc = function (var pfc: THTTP_FILTER_CONTEXT;
    Notificationtype: DWORD; pvNotification: Pointer): DWORD stdcall;

  // The TerminateFilter is an entry point exposed by ISAPI filters. This
  // function indicates to the filter that it is going to be removed from
memory.
  // When this function is called, you should make sure that the filter
closes
  // any attachments it has made to system resources.
```

```
  TTerminateFilter = function (dwFlags: DWORD): BOOL stdcall;

  // Interface structure for an isapi filter
  IISAPIFilter = interface
    function GetFilterVersion(var pVer: HTTP_FILTER_VERSION): BOOL; stdcall;
    function HttpFilterProc(var pfc: THTTP_FILTER_CONTEXT; NotificationType:
DWORD; pvNotification: Pointer): DWORD; stdcall;
    function TerminateFilter(dwFlags: DWORD): BOOL; stdcall;
  end;

implementation

end.
```

Autor: Udo Schmal, veröffentlicht: 29.05.2012, letzte Änderung: 29.08.2024