Writing an ISAPI Filter

Revision 0.5 (DRAFT)

XXX 0, 0000

Overview

This specification describes the basics of writing an ISAPI (Internet Server API) Filter for the Microsoft Internet Information Server. An ISAPI filter is a replaceable dynamic link library (DLL) the server calls on every HTTP (Hypertext Transfer Protocol) request. When the filter is loaded it tells the server what sort of notifications it is interested in. After that, whenever the selected events occur, the filter is called and given the opportunity to process that event.

ISAPI filters are powerful enough to allow for the following applications:

1. Custom authentication schemes
2. Compression
3. Encryption
4. Logging
5. Traffic analysis or other request analysis (looking for requests to "..\..\etc\password" for example)

Multiple filters can be installed. The notification order is based on the priority specified by the filter and then the load order in the registry for any ties.

Note once a filter has expressed interest in a request, it will receive that data regardless of whether the request is for a file, a CGI (Common Gateway Interface, a common HTTP server extension) application or an ISAPI application.

About

ISAPI filters can be used to enhance the Microsoft Internet Information Server with custom features such as enhanced logging of HTTP requests, custom encryption and compression schemes, or new authentication methods. The filter applications sit between the network connection to the clients and the HTTP server. Depending on the options that the filter application chooses, it can act on several server actions, including reading raw data from the client, processing the headers, communications over a secure port (PCT (Personal Communications Technology), SSL (Secure Sockets Layer), etc.), or several other stages in the processing of the HTTP request.

Using ISAPI Filter APIs

Using the ISAPI filter APIs requires that the filter application's path is inserted into the registry. When the server starts up it reads this value and loads the DLLs listed. It then calls the **GetFilterVersion** entrypoint to exchange version information and determine the notifications desired and priority to deliver them. As the events happen, the server will notify each filter application (by calling the **HttpFilterProc** entrypoints) that registered for that event in the priority order requested by **GetFilterVersion**. In the event of a tie, the order listed in the registry is used.

DLL Entry Points

Every ISAPI filter DLL must export at least two entry points. These are **GetFilterVersion** and **HttpFilterProc**. The **GetFilterVersion** entrypoint is passed a **HTTP_FILTER_VERSION** structure that must be filled out with version information, requested events, and priority level. ISAPI filter applications should only register for the events that are required. Registering for extraneous events can have a significant negative impact on performance and scalability. After this exchange, every time the server processes one of the available events it will call any filters that have registered for that event. The order that the server will call the filters depends first on the priority specified in the dwFlags member of **HTTP_FILTER_VERSION** by **GetFilterVersion**. In the event that two or more different filters have registered for the same event at the same priority, the order that the filters were loaded from the registry determines the order that they will be called.

When the **HttpFilterProc** entrypoint is called the filter will typically perform a switch on the NotificationType parameter to determine what action to take. For example, an encryption or compression filter will probably register for reading and writing raw data, while a logging filter will probably only register for the log event. Most filters will also register for the end of net session event. This event is a good time to recycle any buffers used by that client request. For performance reasons most filters will probably keep a pool of filter buffers and only allocate or free when the pool becomes empty or too large, so save on the overhead of the memory management. One useful callback is the **AllocMem** callback in the **HTTP_FILTER_CONTEXT** structure. This will allocate memory that is automatically freed when the communication with the client is terminated. As noted, this can have a negative impact on performance, but with careful use it can be a valuable tool.

Functions

GetFilterVersion

BOOL WINAPI GetFilterVersion(

PHTTP_FILTER_VERSIO
N *pVer*

);

Parameters

pVer
The **HTTP_FILTER_VERSION** structure pointed to by this parameter contains version information for the server and fields for the client to indicate version number, notifications and priority desired. There is also a space for the fiilter application to register a small description of itself.

Return Value

The return code indicates if the filter was properly loaded. If the filter returns FALSE then the filter application will be unloaded and it will not receive any notifcations.

Remarks

This API, implemented in the ISAPI filter application, is the first entrypoint called by the Internet Information Server. It is important that only the necessary notifications are specified in the pVer->dwFlags member. Some notifications can have a strong impact on performance and scalability. In addition to the notification flags described in the **HttpFilterProc** documentation, there are also priority flags to specify which order to call the filter:

SF_NOTIFY_ORDER_DEFAULT
Load the filter at the default priority (recommended)

SF_NOTIFY_ORDER_LOW
Load the filter at the low priority.

SF_NOTIFY_ORDER_MEDIUM
Load the filter at a medium priority.

SF_NOTIFY_ORDER_HIGH
Load the filter at a high priority.

See Also

HttpFilterProc, **HTTP_FILTER_VERSION**

HttpFilterProc

DWORD WINAPI HttpFilterProc(
PHTTP_FILTER_CONTEXT *pfc*,

DWORD *NotificationType*,

LPVOID *pvNotification*

);

Parameters

pfc
The **HTTP_FILTER_CONTEXT** structure pointed to by this parameter contains context information. The pFilterContext member can be used by the filter to associate any context information with the HTTP request. The **SF_NOTIFY_END_OF_NET_SESSION** notification can be used to release any such context information.

NotificationType
Indicates the type of event being processed. Valid types are:

SF_NOTIFY_SECURE_PORT
Notify application only for sessions over a secure port

SF_NOTIFY_NONSECURE_PORT
Notify application only for sessions over a secure port

SF_NOTIFY_READ_RAW_DATA
Allow the application to see the raw data. The data returned will contain both headers and data

SF_NOTIFY_PREPROC_HEADERS
The server has pre-processed the headers.

SF_NOTIFY_AUTHENTICATION
The server is authenticating the client.

SF_NOTIFY_URL_MAP
The server is mapping a logical URL to a physical path.

SF_NOTIFY_SEND_RAW_DATA
The server is sending raw data back to the client.

SF_NOTIFY_LOG
The server is writing information to the server log.

SF_NOTIFY_END_OF_NET_SESSION
The session with the client is ending.

pvNotification
Notification-specific structure

Notification Type	pvNotification points to
SF_NOTIFY_READ_RAW_DATA SF_NOTIFY_SEND_RAW_DATA	HTTP_FILTER_RAW_DATA
SF_NOTIFY_PREPROC_HEADERS	HTTP_FILTER_PREPROC_HEADERS
SF_NOTIFY_AUTHENTICATION	HTTP_FILTER_AUTHENT
SF_NOTIFY_URL_MAP	HTTP_FILTER_URL_MAP
SF_NOTIFY_LOG	HTTP_FILTER_LOG

Return Code

Indicates how the application handled the event. Possible return codes are:

SF_STATUS_REQ_FINISHED
The filter has handled the HTTP request. The server should disconnect the session.

SF_STATUS_REQ_FINISHED_KEEP_CONN
Same as SF_STATUS_REQ_FINISHED except the server should keep the TCP session open if the option was negotiated

SF_STATUS_REQ_NEXT_NOTIFICATION
The next filter in the notification chain should be called

SF_STATUS_REQ_HANDLED_NOTIFICATION
This filter handled the notification. No other handlers should be called for this particular notification type

SF_STATUS_REQ_ERROR
An error occurred. The server should use **GetLastError** and indicate the error to the client

SF_STATUS_REQ_READ_NEXT
The filter is an opaque stream filter and we're negotiating the session parameters. Only valid for raw read notification.

Remarks

This is where the core work of the ISAPI filter applications is done. The various structures pointed to by pvNotification contain data and function pointers specific to these operations. See the structure details for more information.

See Also

HTTP_FILTER_CONTEXT, HTTP_FILTER_RAW_DATA, HTTP_FILTER_PREPROC_HEADERS, HTTP_FILTER_AUTHENT, HTTP_FILTER_URL_MAP, HTTP_FILTER_LOG

Structures

HTTP_FILTER_VERSION

```
typedef struct _HTTP_FILTER_VERSION
{
DWORD     dwServerFilterVersion;
DWORD     dwFilterVersion;
CHAR      lpszFilterDesc[SF_MAX_FILTER_DESC_LEN+1];
DWORD     dwFlags;
} HTTP_FILTER_VERSION, *PHTTP_FILTER_VERSION;
```

Members

dwServerFilterVersion [IN]
Version of the spec used by the server. The version of the current header file is HTTP_FILTER_REVISION

dwFilterVersion [OUT]
Version of the spec used by the server. The version of the current header file is HTTP_FILTER_REVISION

lpszFilterDesc[OUT]
Location to store a short string description of ISAPI filter application

dwFlags [OUT]
Combination of SF_NOTIFY_* flags to specify what events this application is interested in. See **HttpFilterProc** for a list of valid flags.

Remarks

This structure is passed to the application's **HttpFilterProc** entrypoint by the server.

See Also

HttpFilterProc

HTTP_FILTER_CONTEXT

```
typedef struct _HTTP_FILTER_CONTEXT
{
DWORD    cbSize;
DWORD    Revision;
PVOID    ServerContext;
DWORD    ulReserved;
BOOL     fIsSecurePort;
PVOID    pFilterContext;
BOOL    (WINAPI * GetServerVariable) (
    struct _HTTP_FILTER_CONTEXT *    pfc,
    LPSTR      lpszVariableName,
    LPVOID     lpvBuffer,
    LPDWORD    lpdwSize
    );
BOOL    (WINAPI * AddResponseHeaders) (
    struct _HTTP_FILTER_CONTEXT *    pfc,
    LPSTR    lpszHeaders,
    DWORD    dwReserved
    );
BOOL    (WINAPI * WriteClient)  (
    struct _HTTP_FILTER_CONTEXT *    pfc,
    LPVOID     Buffer,
    LPDWORD    lpdwBytes,
    DWORD      dwReserved
    );
VOID *     (WINAPI * AllocMem) (
    struct _HTTP_FILTER_CONTEXT *    pfc,
    DWORD      cbSize,
    DWORD      dwReserved
    );
BOOL    (WINAPI * ServerSupportFunction) (
    struct _HTTP_FILTER_CONTEXT *    pfc,
    enum SF_REQ_TYPE    sfReq,
    PVOID       pData,
    DWORD       ul1,
    DWORD       ul2
    );
} HTTP_FILTER_CONTEXT, *PHTTP_FILTER_CONTEXT;
```

Members

cbSize [IN]
  Size of this structure, in bytes.

Revision [IN]
  Revision level of this structure. This is less than or equal to the version of the spec, HTTP_FILTER_REVISION.

ServerContext [IN]
  Reserved for server use.

ulReserved [IN]
  Reserved for server use.

fIsSecurePort [IN]

TRUE indicates that this event is over a secure port.

pFilterContext [IN/OUT]

A pointer to be used by the filter for any context information that the filter wants to associate with this request. Any memory associated with this request can be safely freed during the SF_NOTIFY_END_OF_NET_SESSION notification.

```
BOOL    (WINAPI * GetServerVariable) (
```

Pointer to a function to retrieve information about the server and this connection. See the ISAPI application documentation for **GetServerVariable** for details.

```
    struct _HTTP_FILTER_CONTEXT *    pfc,    pfc passed to HttpFilterProc
    LPSTR    lpszVariableName,    Server variable to retrieve.
    LPVOID    lpvBuffer,    Buffer to store value of variable
    LPDWORD    lpdwSize    Size of buffer pointed to lpvBuffer
    );
BOOL    (WINAPI * AddResponseHeaders) (
```

Pointer to a function that adds a header to the HTTP response. See the ISAPI documentation on **ServerSupportFunction**, HSE_SEND_RESPONSE_HEADER for details.

```
    struct _HTTP_FILTER_CONTEXT *    pfc,    pfc passed to HttpFilterProc
    LPSTR    lpszHeaders,    Pointer string containing headers to add.
    DWORD    dwReserved    Reserved for future use. Must be 0.
    );
BOOL    (WINAPI * WriteClient)  (
```

Pointer to a function that sends raw data back to the client. See the ISAPI documentation on **WriteClient** for details.

```
struct _HTTP_FILTER_CONTEXT *    pfc,   pfc passed to HttpFilterProc
    LPVOID    Buffer,                   Buffer containing data to
                                        send to the client.
    LPDWORD    lpdwBytes,               Size of the buffer
                                        pointed to by Buffer.
    DWORD    dwReserved                 Reserved for future use.
    );
VOID *     (WINAPI * AllocMem) (
```

Pointer to a function used to allocate memory. Any memory allocated with this function will automatically be freed when the request is completed.

```
struct _HTTP_FILTER_CONTEXT *    pfc,    pfc passed to HttpFilterProc
    DWORD    cbSize,    Size of the buffer to allocate
    DWORD    dwReserved    Reserved for future use.
    );
BOOL    (WINAPI * ServerSupportFunction) (
```

Pointer to a function used to extend the ISAPI filter APIs. Parameters are specific to the extensions.

```
struct _HTTP_FILTER_CONTEXT *    pfc,
    enum SF_REQ_TYPE    sfReq,
    PVOID    pData,
    DWORD    ul1,
    DWORD    ul2
    );
```

Pointer to a function used to extend the ISAPI filter APIs. Parameters are specific to the extensions. Possible values for sfReq are:

SF_REQ_SEND_RESPONSE_HEADER
: 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'.

Parameters

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_ADD_HEADERS_ON_DENIAL
: 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 without 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.

Parameters

pData
: Zero terminated string pointing to one or more header lines with terminating '\r\n'.

SF_REQ_SET_NEXT_READ_SIZE
: Only used by raw data filters that return SF_STATUS_READ_NEXT.

Parameters

ul1
: size in bytes for the next read.

HTTP_FILTER_RAW_DATA

```
typedef struct _HTTP_FILTER_RAW_DATA
{
PVOID       pvInData;
DWORD       cbInData;
DWORD       cbInBuffer;
DWORD       dwReserved;
} HTTP_FILTER_RAW_DATA, *PHTTP_FILTER_RAW_DATA;
```

Members

pvInData [IN]
Pointer to the data buffer (input or output).

cbInData [IN]
Amount of data in the buffer pointed to by pvInData.

cbInBuffer [IN]
Size of the buffer pointed to by pvInData.

dwReserved [IN]
Reserved for future use.

Remarks

This structure is passed to the SF_NOTIFY_READ_RAW_DATA and SF_NOTIFY_SEND_RAW_DATA notification routines.

See Also

HttpFilterProc

HTTP_FILTER_PREPROC_HEADERS

```
typedef struct _HTTP_FILTER_PREPROC_HEADERS
{
BOOL    (WINAPI * GetHeader) (
    struct _HTTP_FILTER_CONTEXT *    pfc,
    LPSTR    lpszName,
    LPVOID    lpvBuffer,
    LPDWORD    lpdwSize
    );
BOOL    (WINAPI * SetHeader) (
    struct _HTTP_FILTER_CONTEXT *    pfc,
    LPSTR    lpszName,
    LPSTR    lpszValue
    );
BOOL    (WINAPI * AddHeader) (
    struct _HTTP_FILTER_CONTEXT *    pfc,
    LPSTR    lpszName,
    LPSTR    lpszValue
    );
DWORD    dwReserved;
} HTTP_FILTER_PREPROC_HEADERS, *PHTTP_FILTER_PREPROC_HEADERS;
```

Members

```
BOOL (WINAPI * GetHeader) (
struct _HTTP_FILTER_CONTEXT *    pfc,
LPSTR    lpszName,
LPVOID    lpvBuffer,
LPDWORD    lpdwSize
 );
```

Pointer to a function that retrieves the specified header value. Header names should include the trailing colon (':'). The special values 'method', 'url' and 'version' can be used to retrieve the individual portions of the request line.

pfc
Filter context for this request from he pfc passed to the HttpFilterProc.

lpszName
The name of the header to retrieve.

lpvBuffer
Pointer to a buffer of size lpdwSize where the valueof the header will be stored.

```
BOOL (WINAPI * SetHeader) (
struct _HTTP_FILTER_CONTEXT *    pfc,
LPSTR    lpszName,
LPSTR    lpszValue
);
```

Pointer to a function used to change or delete the value of a header.

pfc
Filter context for this request from he pfc passed to the **HttpFilterProc**.

lpszName
Pointer to the name of the header to change or delete.

lpszValue
Pointer to the string to change the header to, or a pointer to '\0' to delete the header.

```
BOOL (WINAPI * AddHeader) (
struct _HTTP_FILTER_CONTEXT *    pfc,
LPSTR    lpszName,
LPSTR    lpszValue
);
```

Pointer to a function to add a header.

pfc
Filter context for this request from he pfc passed to the **HttpFilterProc**.

lpszName
Pointer to the name of the header to change or delete.

lpszValue
Pointer to the string to change the header to, or a pointer to '\0' to delete the header.

Remarks

This structure is pointed to by the pvNotification in the **HttpFilterProc** when NotificationType is SF_NOTIFY_PREPROC_HEADERS when the server is about to process the client headers

See Also

HttpFilterProc

HTTP_FILTER_AUTHENT

```
typedef struct _HTTP_FILTER_AUTHENT
{
CHAR *    pszUser;
DWORD    cbUserBuff;
CHAR *    pszPassword;
DWORD    cbPasswordBuff;
} HTTP_FILTER_AUTHENT, *PHTTP_FILTER_AUTHENT;
```

Members

pszUser [IN/OUT]
Pointer to a string containing the username for this request. An empty string indicates an anonymous user.

cbUserBuff [IN]
Size of the buffer pointed to by pszUser. This is guaranteed to be at least SF_MAX_USERNAME.

pszPassword [IN/OUT]
Pointer to a string containing the password for this request.

cbPasswordBuff [IN]
Size of the buffer pointed to by pszPassword. This is guaranteed to be at least SF_MAX_PASSWORD.

Remarks

This structure is pointed to by the pvNotification in the **HttpFilterProc** when NotificationType is SF_NOTIFY_AUTHENTICATION when the server is about to authenticate the client. This can be used to implement a different authentication scheme.

See Also

HttpFilterProc

HTTP_FILTER_URL_MAP

```
typedef struct _HTTP_FILTER_URL_MAP
{
const CHAR *    pszURL;
CHAR *          pszPhysicalPath;
DWORD           cbPathBuff;
} HTTP_FILTER_URL_MAP, *PHTTP_FILTER_URL_MAP;
```

Members

pszURL [IN]
Pointer to the URL that is being mapped to a physical path.

pszPhysicalPath [IN/OUT]
Pointer to the buffer where the physical path is stored.

cbPathBuff [IN]
Size of the buffer pointed to by pszPhysicalPath.

Remarks

This structure is pointed to by the pvNotification in the **HttpFilterProc** when NotificationType is SF_NOTIFY_URL_MAP when the server is about to map the specified URL to a physical path. Filters can modify the physical path in place.

See Also

HttpFilterProc

HTTP_FILTER_LOG

```
typedef struct _HTTP_FILTER_LOG
{

const CHAR *    pszClientHostName;
const CHAR *    pszClientUserName;
const CHAR *    pszServerName;
const CHAR *    pszOperation;
const CHAR *    pszTarget;
const CHAR *    pszParameters;
DWORD        dwHttpStatus;
DWORD        dwWin32Status;
} HTTP_FILTER_LOG, *PHTTP_FILTER_LOG;
```

Members

pszClientHostName [IN/OUT]
Client's host name.

pszClientUserName [IN/OUT]
Client's user name.

pszServerName [IN/OUT]
Name of the server the client connected to.

pszOperation [IN/OUT]
HTTP command.

pszTarget [IN/OUT]
Target of the HTTP command.

pszParameters [IN/OUT]
Parameters passed to the HTTP command.

dwHttpStatus [IN/OUT]
HTTP return status.

dwWin32Status [IN/OUT]
Win32 Error code.

Remarks

This structure is pointed to by the pvNotification in the **HttpFilterProc** when NotificationType is SF_NOTIFY_LOG when the server is about to log information to the server log file. The strings cannot be changed but pointers can be replaced. If the string pointers are changed, the memory they point to must remain valid until the next notification.

The DLL Entry Points

Every filter is contained in a separate DLL with 2 common entrypoints. These are **GetFilterVersion** and **HttpFilterProc**. When the DLL is loaded **GetFilterVersion** is called which lets the filter know the version of the server and allows the filter to tell the server the version of the filter and the events that the filter is interested in. After this the server will call the filter's **HttpFilterProc** entrypoint with appropriate notifications. Note that filters should only register for notifications that the filter needs to see - some filter notifications are very expensive in terms of CPU resources and I/O throughput and can have a dramatic effect on the speed and scalability of the Microsoft Internet Information Server.

```
BOOL
WINAPI
GetFilterVersion(
    HTTP_FILTER_VERSION * pVer
    );

DWORD
WINAPI
HttpFilterProc(
    HTTP_FILTER_CONTEXT *      pfc,
    DWORD                      NotificationType,
    VOID *                     pvNotification
    );
```