

```

/*
-----
*
*   File:      dsapi3.c
*   Author:    Ed Wrenbeck based on original work by Chris Stead
*
*
*-----
*/

#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include "dsapifilter.h"
#include <time.h>

#if !defined(DLLEXPORT)
#ifdef WIN32
#define DLLEXPORT __declspec(dllexport)
#else
#define DLLEXPORT
#endif
#endif

char * filter_name = "Adjust HTTP Headers";
char * version_number = "1.08";

#define TRUE    1
#define FALSE   0

int      debug=TRUE;                // turn on debugging

// Function prototypes for HTTP header handling functions.
unsigned int Response( FilterContext* context, FilterResponse* response);
void setHTTPHeader( FilterContext* context, FilterResponse* response, char *
pszHeaderName, char * pszHeaderDetails);

//
*****
*****
// Filter initialisation
//
*****
*****
DLLEXPORT int FilterInit(FilterInitData* filterInitData)
{
    char strTemp[128];
    time_t ltime;
    struct tm *tmToday;

    // Set up the filter to respond to HTTP responses
    filterInitData->appFilterVersion = kInterfaceVersion;
    filterInitData->eventFlags = kFilterResponse;

    // Name the filter
    strcpy(filterInitData->filterDesc, filter_name);

    // Send a filter initialisation message to the Domino server console.
    time( &ltime);
    tmToday = localtime( &ltime );
    strftime( strTemp, sizeof( strTemp ), "%d/%m/%Y %H:%M:%S", tmToday );
    printf( "%s  DSAPI Filter: %s version %s started\n", strTemp,
filter_name, version_number);

    return kFilterHandledEvent;
} // end FilterInit

//
*****

```

```

*****
// Filter Termination
//
*****
*****
DLLEXPORT unsigned int TerminateFilter(unsigned int reserved)
{
    char strTemp[128];
    time_t ltime;
    struct tm *tmToday;

    // Send a filter termination message to the Domino server console.
    time( &ltime);
    tmToday = localtime( &ltime );
    strftime( strTemp, sizeof( strTemp ), "%d/%m/%Y %H:%M:%S", tmToday );
    printf( "%s  DSAPI Filter: %s shutdown\n",strTemp, filter_name);

    return kFilterHandledEvent;
} // end TerminateFilter

//
*****
*****
// Filter processing
//
*****
*****
DLLEXPORT int HttpFilterProc( FilterContext* context, unsigned int eventType,
void * eventData)
{
    // Execute filter processing according to the filter event.
    switch (eventType)
    {
        case kFilterResponse:
            return Response( context, eventData );

        default:
            return kFilterNotHandled;
    }
} // end HttpFilterProc

//
*****
*****
// Handle Response
//
*****
*****
unsigned int Response( FilterContext* context, FilterResponse* response)
{
    FilterRequest request;
    unsigned int uiHeaderLength;
    char * pPos;
    char szURL[ 1024 ];
    char szHeader[ 1024 ];
    unsigned int uiError;
    int fHeadersRequired = FALSE;

    context->GetRequest( context, &request, &uiError );

    if ( uiError == 0 )
    {
        // Convert the URL to lower case
        strcpy( szURL, request.URL );
        for( pPos = szURL; *pPos; pPos++ )
            *pPos = tolower( *pPos );
    }
}

```

```

// Check for URLs to add headers to (strings must be in lower case).
// *****
if (strstr(szURL, "mail") == NULL)
{
    fHeadersRequired = FALSE;
}
else
{
    fHeadersRequired = TRUE;
}

// Tell the browser not to cache anything
// fHeadersRequired = TRUE;

// Tell the browser to cache everything
// fHeadersRequired = FALSE;

// Add headers
// *****
if (fHeadersRequired == TRUE)
{
    fHeadersRequired = FALSE;
    // Determine what the response type is (e.g. HTML, image, etc.
    uiHeaderLength = response->GetHeader( context, "Content-Type",
    szHeader, sizeof(szHeader), &uiError);

    if ( uiHeaderLength != 0 )
    {
        // Response is text
        if ( strstr( szHeader, "text/" ) )
        {
            fHeadersRequired = TRUE;
        }
        // Response is an image
        else
        {
            if ( strstr( szHeader, "image/" ) )
            {
                fHeadersRequired = TRUE;
            } // end if ( strstr( szBuffer, "image/" ) )
        } // end else
    } // end if ( uiHeaderLength != 0 )

    // Add cache control headers...
    if ( fHeadersRequired == TRUE )
    {
        setHTTPHeader(context, response, "Cache-Control",
        "no-cache, must-revalidate, proxy-revalidate");
        setHTTPHeader(context, response, "Expires",
        "Mon, 01 Jan 1990 12:00:00 GMT");
        // printf("\nNo Cache: %s", szURL);
    } // end if ( fHeadersRequired == TRUE )
    //else
    //{
    // printf("\nCached: %s", szURL);
    //}
} // end if ( fHeadersRequired )
//else
//{
// printf("\nCached: %s", szURL);
//}
} // end if ( uiError == 0 )

```

```
    return kFilterNotHandled;
}

//
*****
*****
// Set the HTTP header
//
*****
*****
void setHTTPHeader( FilterContext* context, FilterResponse* response, char *
pszHeaderName, char * pszHeaderDetails)
{
    unsigned int uiHeaderLength;
    unsigned int uiError;
    char szBuffer[ 1024 ];

    // Retrieve the HTTP header, if it exists then update it,
    // otherwise create a new new header.
    uiHeaderLength = response->GetHeader(context, pszHeaderName, szBuffer,
sizeof(szBuffer), &uiError);

    strcpy(szBuffer, pszHeaderName);
    strcat(szBuffer, ": ");
    strcat(szBuffer, pszHeaderDetails);
    strcat(szBuffer, "\n");

    if (uiHeaderLength == 0)
    {
        response->AddHeader(context, szBuffer, &uiError);
    }

    return;
}
```