Server Modules and Filters

Basic Definitions

- *A server API program is a program which is written to tightly link into a Web server and runs inside the server process. Given this coupling these programs can access the various stages of an HTTP request as it is received, handled and returned*

- *Server API programs are very specific to the server they are running on and are generally named for the particular server they work with*
  - *Ex: Apache Modules, ISAPI Extensions, ISAPI Filters, NSAPI plugins, etc.*

Motivations

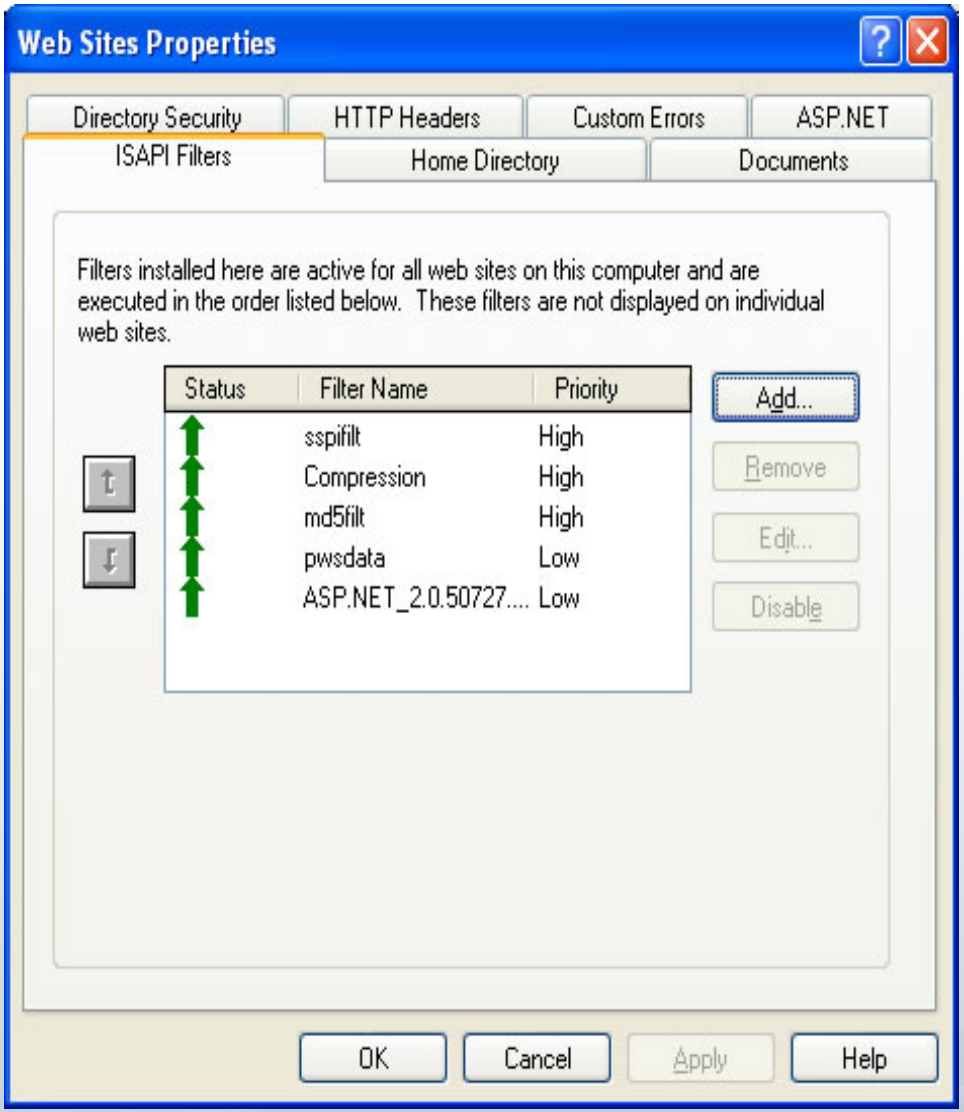
- The reasons you would want to use a server API program are two fold
  1. You need very low level access to the request and response stream to perform some useful function such as transformation
  2. You need the performance that this programming model provides
- The reasons you'd want to avoid this aproach
  1. Complexity
  2. Safety

Triggers

- Some server API modules are transparent and will be put in some stage of the request/response lifecycle looking at all the data coming by
  - Such programs are generally called filters and there may be a number of them modifying some or all requests and responses in a particular defined order
- Others are trigger by some external indicator generally path (URI requested) or file extension (ex. content handlers)
  - Filters of course can be written to select what to operate on by such triggers as well
  - BTW server-side frameworks generally implemented as extensions, filters or a combo of both! (ex. ASP.NET, PHP, etc.)

Filter Order

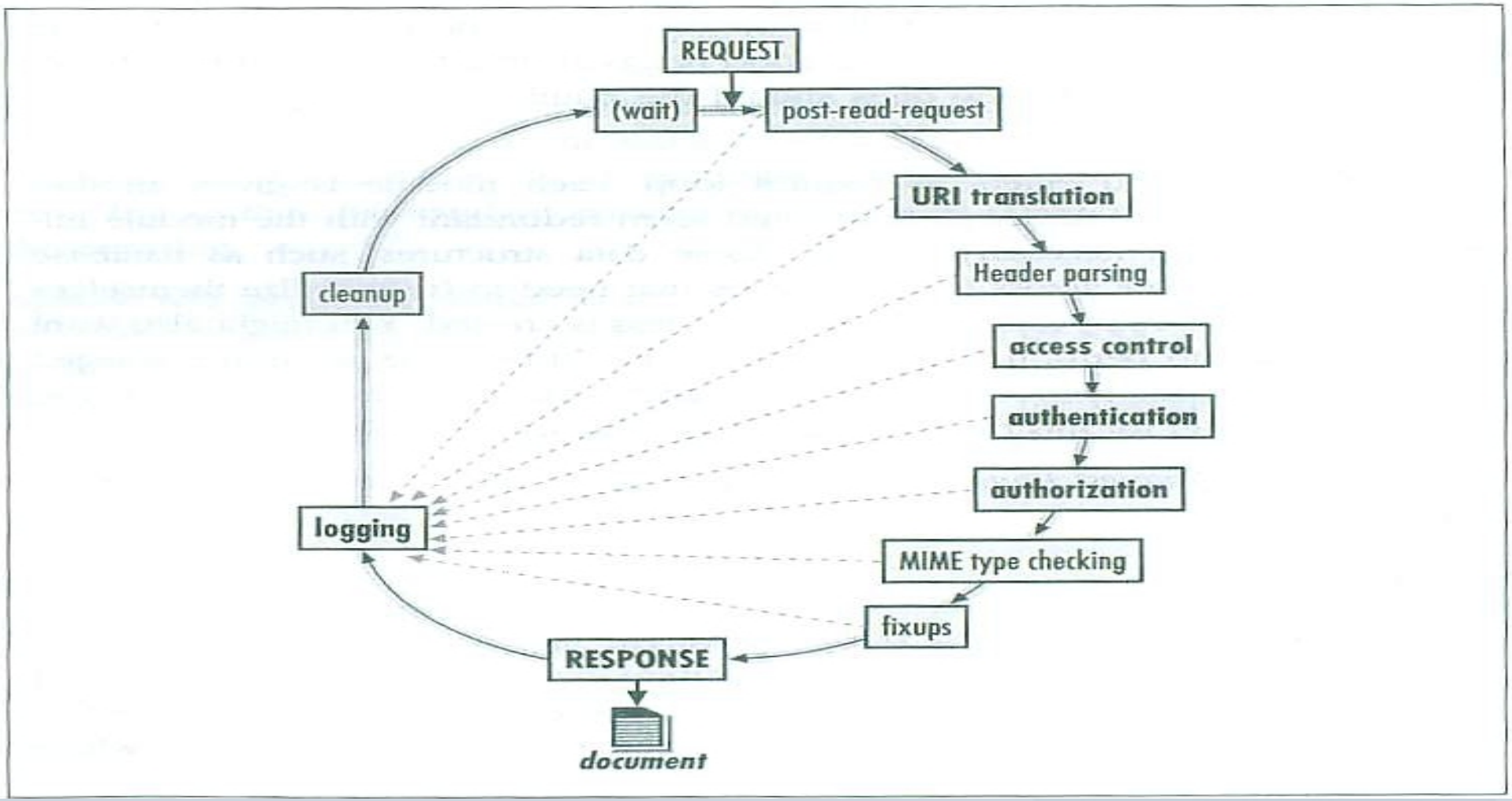
- Given that filters often change content the order to which they are applied is pretty important.
- As an example look at how IIS prioritizes various filters



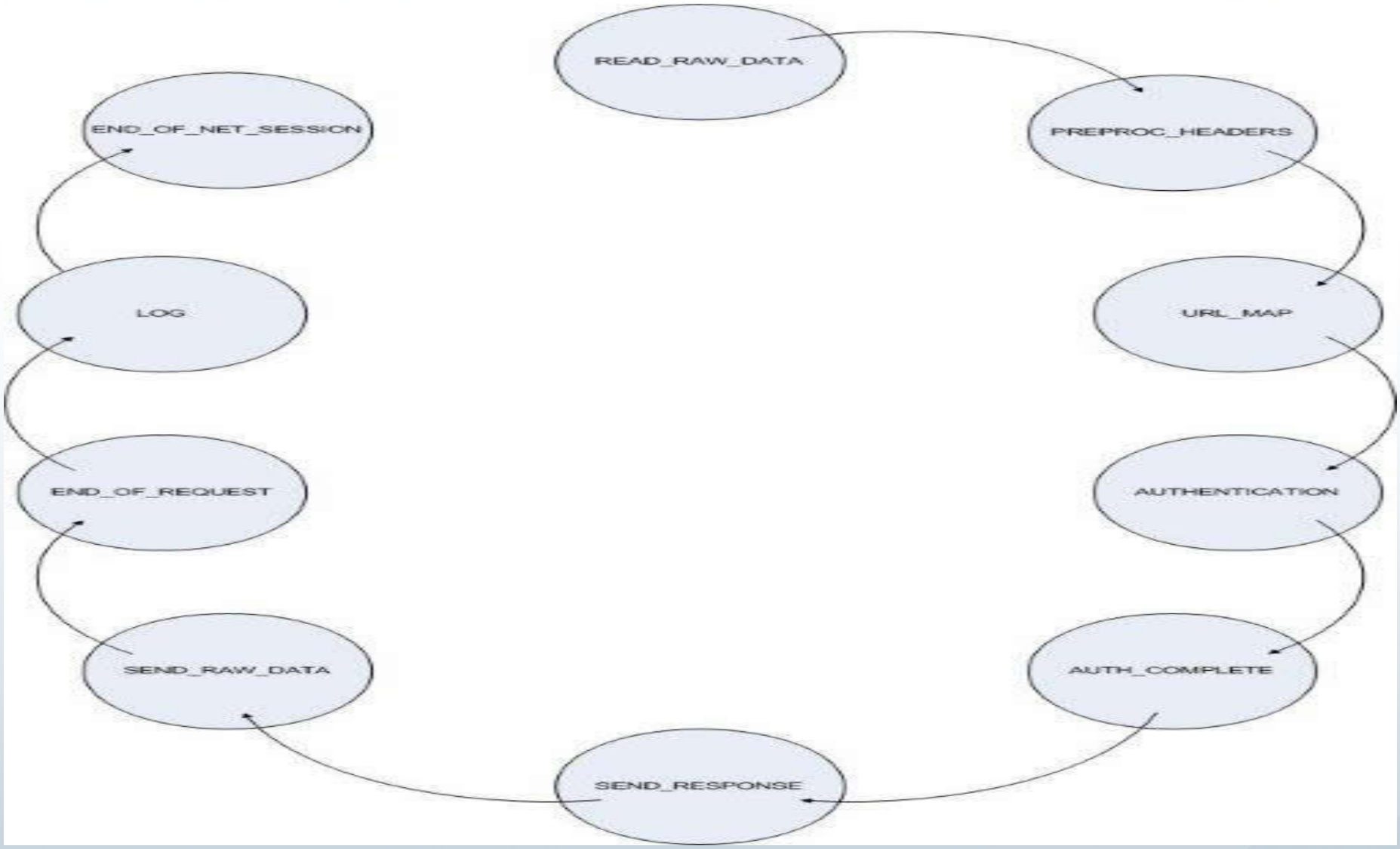
Example Uses of Server API Programs

- Frameworks
- Performance
  - Compression – dynamically intercepting the data stream and running a gzip on it
  - Injecting headers to improve cachability
  - Re-sampling or reformatting content for improved delivery
- Security
  - Monitoring and dropping inbound requests before they get to the application layer
  - Removing dangerous content or scrubbing data on the response
  - Custom authentication (ex. talk to a DB for SSO support)
- Transformation
  - Rewrite URLs or content for security, usability, localization, and so on
- Logging
  - Change the format of what is recorded

Apache Request / Response Lifecycle



IIS Request / Response Lifecycle



Sample Code - ISAPI

```
#include <windows.h>
#include <httpfilt.h>

#define FILENAME "getie8.htm"


BOOL WINAPI GetFilterVersion (PHTTP_FILTER_VERSION pFilterVersion)
{
   pFilterVersion->dwFilterVersion = HTTP_FILTER_REVISION;

   strcpy (pFilterVersion->lpszFilterDesc,"Smart Redir Filter");
   pFilterVersion->dwFlags= (     SF_NOTIFY_ORDER_HIGH     |
                                                  SF_NOTIFY_SECURE_PORT     |
                                                  SF_NOTIFY_NONSECURE_PORT |
                                                  SF_NOTIFY_URL_MAP

                                          );

  return TRUE;
}
```

Sample Code - ISAPI

```
DWORD WINAPI HttpFilterProc (PHTTP_FILTER_CONTEXT pFC,
                        DWORD dwNotificationType,
                        LPVOID pvNotification)
{
   HTTP_FILTER_URL_MAP *pUrlMap;
   CHAR  szTemp [256], *lpszUserAgent, *lpszServer;
   DWORD dwSize = 0;

         if (dwNotificationType == SF_NOTIFY_URL_MAP)
   {
      pUrlMap = (HTTP_FILTER_URL_MAP *) pvNotification;
                  if (strstr (strlwr ((CHAR *)pUrlMap->pszURL), "ieonly") )
                  {
                           // This call should fail w/error ERROR_INSUFFICIENT_BUFFER
                           //  you should check, for other errors.
                           pFC->GetServerVariable (pFC,"HTTP_USER_AGENT",
                               lpszUserAgent,&dwSize);
                           lpszUserAgent = (CHAR *) pFC->AllocMem (pFC,dwSize,0);
```

Sample Code - ISAPI

```
// request is to IEONLY directory, check user agent
        if (!pFC->GetServerVariable (pFC,"HTTP_USER_AGENT",
                        lpszUserAgent,&dwSize))
                {
                        pFC->ServerSupportFunction (pFC, SF_REQ_SEND_RESPONSE_HEADER,
                                                (PVOID) "200 OK", 0,0);
            wsprintf (szTemp, "GetServerVariable failed: %d",
                    GetLastError());
            dwSize = lstrlen (szTemp);
            pFC -> WriteClient (pFC, szTemp, &dwSize, 0);
                    return SF_STATUS_REQ_FINISHED;
                }
if (!strstr (strlwr (lpszUserAgent), "msie 8") )
        {
            // not an IE 8.0 browser, redirect client
            // Get the name of the server. But first determine size
                        // of the buffer.
```

Sample Code - ISAPI

```
                    dwSize = 0;
                    pFC->GetServerVariable (pFC,"SERVER_NAME",
                lpszServer,&dwSize);
                    lpszServer = (CHAR *) pFC->AllocMem (pFC,dwSize,0);
        if (!pFC->GetServerVariable (pFC,"SERVER_NAME",
                    lpszServer,&dwSize))

                        {
                            pFC->ServerSupportFunction (pFC,
SF_REQ_SEND_RESPONSE_HEADER,
                                                    (PVOID) "200 OK", 0,0);
                wsprintf (szTemp, "GetServerVariable failed: %lu", GetLastError());
                            dwSize = lstrlen (szTemp);
                            pFC -> WriteClient (pFC, szTemp, &dwSize,
0);
                            return SF_STATUS_REQ_FINISHED;
                        }

                        wsprintf (szTemp, "Location: http://%s/%s/\r\n\r\n", lpszServer,
FILENAME);
```

Sample Code - ISAPI

```
                    pFC->ServerSupportFunction (pFC, SF_REQ_SEND_RESPONSE_HEADER,
                                    (PVOID) "302 Redirect",
                                                    (DWORD) szTemp,0);
                    return SF_STATUS_REQ_FINISHED;

                }
            }
        }
        return SF_STATUS_REQ_NEXT_NOTIFICATION;
}
```

Apache Source Example

```
#include <httpd.h>
#include <http_protocol.h>
#include <http_config.h>

static int helloworld_handler(request_rec* r) {
    if (!r->handler || strcmp(r->handler, "helloworld"))   return DECLINED;
    if (r->method_number != M_GET)       return HTTP_METHOD_NOT_ALLOWED;
    ap_set_content_type(r, "text/html;charset=ascii");
    ap_rputs("<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN">n", r);
    ap_rputs("<html><head><title>Hello World!</title></head>;", r);
    ap_rputs("<body><h1>Hello World!</h1></body></html>", r);
    return OK;
}
static void register_hooks(apr_pool_t* pool) {
    ap_hook_handler(helloworld_handler, NULL, NULL, APR_HOOK_MIDDLE);
}
module AP_MODULE_DECLARE_DATA helloworld_module = {
    STANDARD20_MODULE_STUFF,NULL,NULL,NULL,NULL,NULL,
    register_hooks
};
```

http://www.cb1inc.com/2007/12/24/creating-a-hello-world-apache-module-with-kdevelop-on-ubuntu/

Moving away from Server APIs?

- Given the trade-offs people want to avoid direct use of Server API programs
- Some environments try to "split the difference" and address these trade-offs
  - Java servlet and .NET HTTP Handlers/Modules
  - These approaches do increase safety by process isolation but they decrease speed as any other encapsulation would
  - It is debatable if they are really any less complex to write
- Conclusion – even in latest Web servers and programming models server APIs still make sense for some things (consider that even these systems to make it easier are typically are implemented as server API programs themselves!)