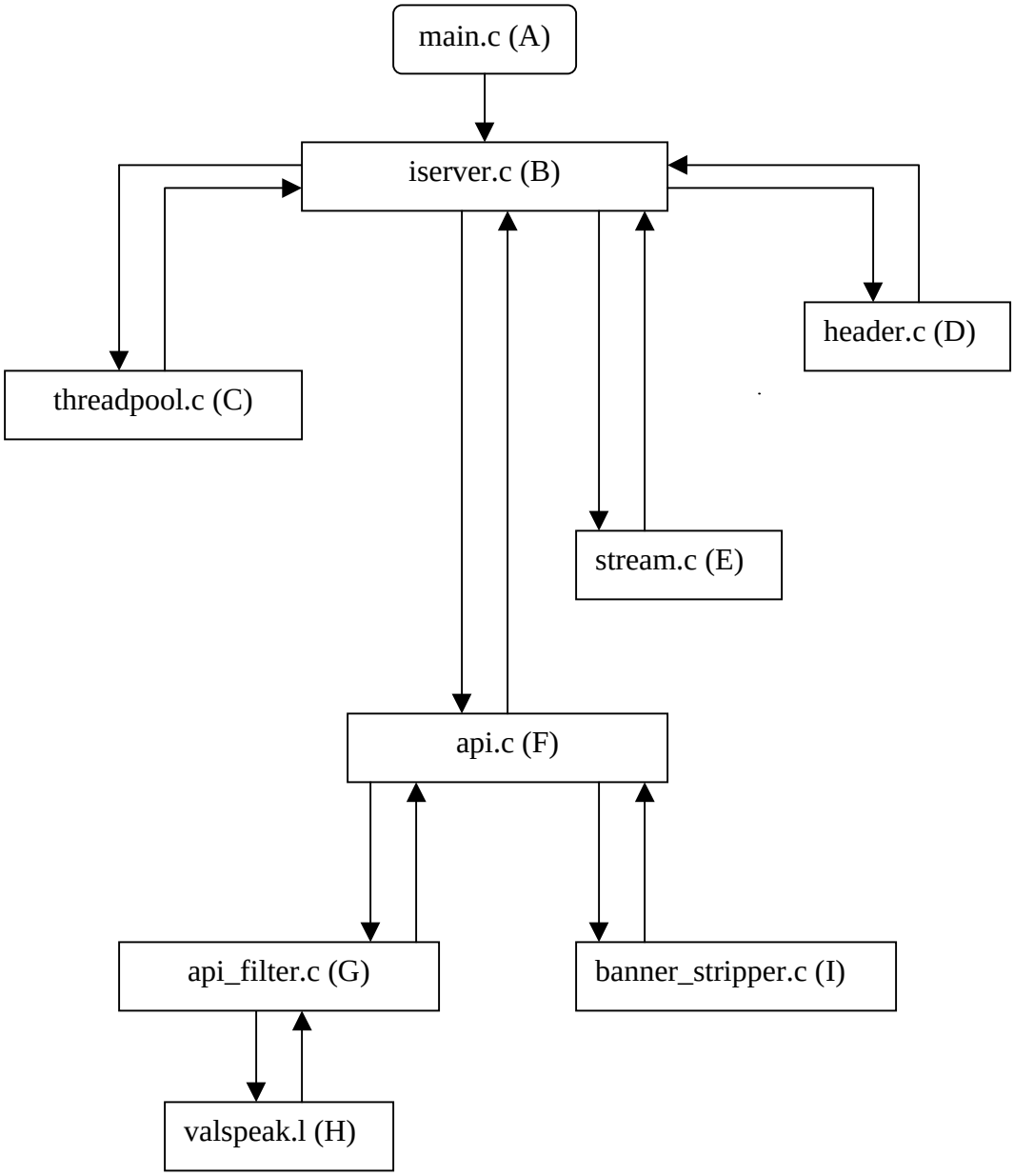
Internet Content Adaptation Protocol
Server Logical Flow Sequence
Version 1 (1) August 28, 2001
©September 11, 2001



A. main.c:

```
/*
 * $Id: //depot/prod/ontap/main/tools/icap/iserver10/main.c#6 $
 *
 * Copyright (c) 2001 Network Appliance, Inc.
 * All rights reserved.
 */

#include "iserver.h"
#include <stdio.h>
#include <string.h>

int
main(int argc, char **argv)
{
      char usage[] = "usage: iserver [-p <portnum>] "
            "[-v] [-te] [-t <num_threads>] [-cp
<reqmod_cache_percent>]"
            " [-mp <modify_percent>]\n\n";
      cinfo.port = DEFAULT_PORT;
      cinfo.num_thr = DEFAULT_NUM_THREADS;
      cinfo.verbose = 0;
      cinfo.trailer_support = 0;
      cinfo.reqmod_cache_percent = DEFAULT_REQMOD_CACHE_PERCENT;
      cinfo.modify_percent = DEFAULT_MODIFY_PERCENT;

      /* parse any command line config stuff here */
      if (argc == 1) {
            fprintf(stderr,
                  "%s",
                  usage);
      } else {
            int i = 1;
            while (i < argc) {
                  if (!strcmp(argv[i], "-p")) {
                        sscanf(argv[i+1], "%d", &cinfo.port);
                        i++;
                  } else if (!strcmp(argv[i], "-v")) {
                        cinfo.verbose = 1;
                  } else if (!strcmp(argv[i], "-t")) {
                        sscanf(argv[i+1], "%d", &cinfo.num_thr);
                        i++;
                        cinfo.num_thr = (cinfo.num_thr > 0) ?
                              cinfo.num_thr : DEFAULT_NUM_THREADS;
                  } else if (!strcmp(argv[i], "-te")) {
                        cinfo.trailer_support = 1;
                  } else if (!strcmp(argv[i], "-cp")) {
                        sscanf(argv[i+1],
                               "%d",
                               &cinfo.reqmod_cache_percent);
                        i++;
                  } else if (!strcmp(argv[i], "-mp")) {
                        sscanf(argv[i+1],
                               "%d",
```

```
                                &cinfo.modify_percent);
                        i++;
                }
                else {
                        fprintf(stderr, "%s", usage);
                        exit(-1);
                }
                i++;
        }
    }

    iserver_reg_api();

    iserver_await_connections();

    srand(getpid());

    return 0;
}
```

B. iserver.c:

```
/*
 * $Id: //depot/prod/ontap/main/tools/icap/iserver10/iserver.c#24 $
 *
 * Copyright (c) 2001 Network Appliance, Inc.
 * All rights reserved.
 */

#include "iserver.h"
#include "threadpool.h"
#include "gbuf.h"
#include "header.h"
#include "stream.h"
#include "api.h"

#include <stdio.h>
#include <string.h>
#include <stdarg.h>
#include <sys/types.h>
#include <assert.h>
#include <signal.h>


#ifdef WINDOWS
#include <Windows.h>
#include <time.h>
#include <errno.h>
#else
#ifdef COMPILE_SOLARIS
#include <errno.h>
#else
#include <sys/errno.h>
#endif
#include <netinet/tcp.h>
#include <sys/time.h>
#include <unistd.h>
#include <semaphore.h>
#endif

/*
 * GLOBALS
 */
#ifdef WINDOWS
static HANDLE time_mtx;
extern HANDLE available_sem;
#else
static pthread_mutex_t time_mtx;
extern sem_t available_sem;
#endif

/*
 * PROTOTYPES
 */
static int iserver_parse_body(iserver_conn_t *conn);
static iserver_conn_t *iserver_alloc_conn(void);
```

```
static void iserver_free_conn(iserver_conn_t *conn);
static void iserver_reset_conn(iserver_conn_t *conn);

/*
 * FUNCTIONS
 */
void
iserver_reg_api(void)
{
        FILE *fp;
      char line;

        /* set function pointers from api */
      iserver_options_response = api_options_response;
      iserver_log_request = api_log_request;
      iserver_preview_reqmod = api_preview_reqmod;
      iserver_preview_respmod = api_preview_respmod;
      iserver_modify_reqmod = api_modify_reqmod;
      iserver_modify_respmod = api_modify_respmod;
#ifdef WINDOWS
      file_mtx = CreateMutex(NULL, FALSE, NULL);
#else
      pthread_mutex_init(&file_mtx, 0);
#endif
      options_str = (char*) malloc(2048*sizeof(char));
      memset(options_str, '\0', 2048*sizeof(char));
      if ((fp = fopen("options.conf", "r" )) == NULL) {
            printf("Error opening options.conf file");
            exit(-1);
      }
      for(line=(char)getc(fp); line!= EOF; line=(char)getc(fp)) {
            strncpy(options_str+strlen(options_str), &line, 1);
      }
      fclose(fp);

        /* get the hostname for via */
      hostname = (char*) malloc(2048*sizeof(char));
      memset(hostname, '\0', 2048*sizeof(char));
      assert(!gethostname(hostname, 2048));
}

void
iserver_await_connections(void)
{
      iserver_conn_t *lconn;
      int allow_addr_reuse=1, temp=0, no_delay=1;
#ifdef WINDOWS
      SOCKET srv_socket;
      WSADATA wsaData;
      if ((WSAStartup(MAKEWORD(2,2), &wsaData)) != 0)
            printf("Winsock init error\n");
      time_mtx = CreateMutex(NULL, FALSE, NULL);
      cinfo.mtx = CreateMutex(NULL, FALSE, NULL);
#else
      int srv_socket;
      pthread_mutex_init(&time_mtx, 0);
      pthread_mutex_init(&cinfo.mtx, 0);
```

```
#endif

      if (cinfo.modify_percent)
            iserver_logmsg(0, "ICAP content modifying server\n");
      else
            iserver_logmsg(0, "ICAP content echoing server\n");

      /* initialize threadpool */
      if (threadpool_init(cinfo.num_thr,
                  (void *(*)(void *))iserver_connected) < 0) {
            iserver_logmsg(0, "Could not allocate threadpool\n");
            goto error;
      }

      iserver_logmsg(0, "Pre-created %d threads\n", cinfo.num_thr);

      /* NOTE: SIGINT does not work for windows */
      /* register the signal handler*/
      if ((*signal)(SIGINT, (void (*)(int))threadpool_signal_handler)
== SIG_ERR) {
            printf("registering ctrl-C sig handler");
            exit(-1);
      }

      /* ignore SIGPIPE to prevent iserver aborts in linux */
      if (sigignore(SIGPIPE) == -1) {
            printf("registering SIGPIPE sig handler");
            exit(-1);
      }
      /* setup listen socket */
      lconn = iserver_alloc_conn();
      if (!lconn)
            goto error;
      assert(lconn->content->blk_size);

      /* get socket */
      srv_socket = socket(AF_INET, SOCK_STREAM, 0);
#ifdef WINDOWS
      if (srv_socket == INVALID_SOCKET) {
            fprintf(stderr,"socket() failed with error
%d\n",WSAGetLastError());
            goto error;
      }
#else
      if (srv_socket < 0) {
            printf("Socket Error");
            goto error;
      }
#endif
      /*
       * allow server-side addresses to be reused (don't have
       * to wait for that annoying timeout)
       */
      temp = setsockopt(srv_socket,
                  SOL_SOCKET,
                  SO_REUSEADDR,
                  (void *)&allow_addr_reuse,
```

```
                    sizeof(allow_addr_reuse));
#ifdef WINDOWS
      if (temp != 0) {
            printf("setsockopt error");
            goto error;
      }
#else
        if (temp < 0)  {
            printf("setsockopt error");
            goto error;
      }
#endif
      /*
       * turn off the Nagle's algorithm
       */
      temp = setsockopt(srv_socket,
                    IPPROTO_TCP,
                    TCP_NODELAY,
                    (void *)&no_delay,
                    sizeof(no_delay));
#ifdef WINDOWS
      if (temp != 0) {
            printf("setsockopt error");
            goto error;
      }
#else
        if (temp < 0)  {
            printf("setsockopt error");
            goto error;
      }
#endif
      /* bind unnamed socket */
      lconn->addr.sin_family = AF_INET;
      lconn->addr.sin_addr.s_addr = htonl(INADDR_ANY);
      lconn->addr.sin_port = htons(cinfo.port);

      temp = bind(srv_socket, (struct sockaddr*)&lconn->addr,
                sizeof(lconn->addr));
#ifdef WINDOWS
      if (temp != 0) {
            printf("bind error");
            goto error;
      }
#else
      if (temp < 0) {
            printf("bind error");
            goto error;
      }
#endif
      /* socket needs to listen for connections*/
        temp = listen(srv_socket, 10);

#ifdef WINDOWS
      if (temp != 0) {
            printf("listen error");
            goto error;
      }
```

```
#else
      if (temp < 0) {
            printf("listen error");
            goto error;
      }
#endif
      /* accept connections */
      for (;;) {
            iserver_conn_t *cconn = iserver_alloc_conn();
            cconn->addr_len = sizeof(cconn->addr);

            /* wait for available thread */
#ifdef WINDOWS
            WaitForSingleObject(available_sem, INFINITE);
#else
            sem_wait(&available_sem);
#endif
            PTHREAD_MUTEX_LOCK(cinfo.mtx);
            cinfo.conn_total++;
            cconn->num = cinfo.conn_total;
              PTHREAD_MUTEX_UNLOCK(cinfo.mtx);

            iserver_logmsg(0,
                        "Waiting for connection %d on port %d",
                        cconn->num,
                        cinfo.port);
            cconn->skt = accept(srv_socket,
                           (struct sockaddr*)&cconn->addr,
                           &cconn->addr_len);
#ifdef WINDOWS
            if (cconn->skt == INVALID_SOCKET){
                  printf("accept error");
                  iserver_free_conn(cconn);
                  goto error;
            }
#else
            if (cconn->skt < 0) {
                  printf("accept error");
                  iserver_free_conn(cconn);
                  goto error;
            }
#endif

      threadpool_dispatch( (void *)cconn );
}

  error:
      printf("\nError: Closing Server\n");
      threadpool_shutdown();
      printf("Closing Listen port.\n");
      if (lconn)
            iserver_free_conn(lconn);
#ifdef WINDOWS
      CloseHandle(time_mtx);
      CloseHandle(cinfo.mtx);
      if (srv_socket != INVALID_SOCKET)
            closesocket(srv_socket);
```

```
#else
      pthread_mutex_destroy(&time_mtx);
      pthread_mutex_destroy(&cinfo.mtx);
      if (srv_socket != -1)
            close(srv_socket);
#endif
}

static void
iserver_send_modified_response(iserver_conn_t* conn)
{
      ca_t* response;

      if (conn->icap_req_type == ICAP_REQMOD) {
            /* call reqmod api* with
               1) icap hdr + client hdr + entire post msg body)
            */
            if (!conn->icap_reqmod_post)
                  assert(conn->content->length == 0);

            response = (*iserver_modify_reqmod)(conn->req_hdr->data,
                                        conn->client_hdr->data,
                                        conn->content->data,
                                        conn->content->length);
      }
      else {
            /* call respmod api* with
               1) icap hdr + origin hdr + origin resp hdr +
               entire msg body

               * - api not to care of preview length, just
                     modify body (of any size) on basis of headers
            */
            response = (*iserver_modify_respmod)(conn->req_hdr->data,
                                         conn->client_hdr->data,
                                         conn->origin_hdr->data,
                                         conn->content->data,
                                         conn->content->length);
      }

      assert(response);
      assert(response->beg);
      iserver_writereply(conn, response);
      free(response->beg);
      free(response);
}

/* entry point for all client-handling threads
 */
void
iserver_connected(iserver_conn_t *conn)
{
      int unsigned temp=0;

      /* don't need to lock, as add/sub is idempotent */
      cinfo.conn_active++     ;
```

```
      while (1) {

            /* parse ICAP request header */
            if (header_read_icap(conn) < 0) {
                  goto done;
            }

            if (conn->icap_req_type != ICAP_OPTIONS) {

                  /* parse client request header */
                  if (header_read_client(conn) < 0) {
                        goto done;
                  }

                  if (conn->icap_req_type == ICAP_RESPMOD&&
                        conn->origin_res_hdr_exist) {

                        /* parse origin response headers */
                        if (header_read_origin(conn) < 0) {
                              goto done;
                        }
                  }

                  if ( conn->icap_reqmod_post ||
                       (conn->icap_req_type == ICAP_RESPMOD
                            && conn->origin_res_body_exist)) {

                        /* parse message body for preview (if any) */
                        if (iserver_parse_body(conn) < 0) {
                              goto done;
                        }
                  }
            }
            else {
                  /* send an options response */

                  char* options_reply;

                  /* FUNCPTR*/
                  options_reply = (*iserver_options_response)(conn-
>icap_uri);
                  assert(options_reply);

                    iserver_writeline(conn, options_reply);
                  free(options_reply);
                  goto done;
            }

            if (conn->icap_req_type == ICAP_LOG) {
                  /* LOG request sent by client, no response to be sent
*/

                  /* FUNCPTR*/
                  (*iserver_log_request)(conn->req_hdr->data);
                  goto done;
            }
```

```
            /* call appropriate API function to decide the
               preview response (if any) */
            if (conn->preview >= 0) {
                  char* response;

                  if (conn->icap_req_type == ICAP_REQMOD) {
                        /* call reqmod api with
                           1) icap hdr + client hdr + post msg body
                           (max size = preview)
                           2) preview length
                        */
                        if (!conn->icap_reqmod_post)
                              assert( conn->content == NULL);

                        response = (*iserver_preview_reqmod)(conn-
>req_hdr->data,
                                                             conn->client_hdr-
>data,
                                                             conn->content-
>data,
                                                             conn->content-
>length,
                                                             conn->preview);
                  }
                  else {
                        /* call respmod api with
                           1) icap hdr + client hdr + origin resp hdr +
                           msg body(max size = preview)
                           2) preview length
                        */
                        response = (*iserver_preview_respmod)(conn-
>req_hdr->data,
                                                              conn->client_hdr-
>data,
                                                              conn->origin_hdr-
>data,
                                                              conn->content-
>data,
                                                              conn->content-
>length,
                                                              conn->preview);
                  }

                  /*
                   *  1) send 100 Continue or 204 depending on
                   *     what api returned ( NULL - 100,
                   *      non-NULL - write out data recvd, free mem and
                   *                 close connection
                   *  2) if (total msg body < preview) && (100 continue
was sent)
                   *     send modified hdrs+content (from char*
returned) by
                   *     calling same api as the one used below
                   */

                  if (response) {
                        if (cinfo.verbose) {
```

```
                        iserver_logmsg(conn, "Sending....\n");
                        iserver_logmsg(conn, response);
                    }

                    temp = iserver_writeline(conn, response);
                    free(response);
                    if (temp<0)
                        goto done;
                }
                else {
                    if (cinfo.verbose && !conn->short_preview)
                        iserver_logmsg(conn,
                                    "Sending \"ICAP/1.0 100"
                                    " Continue\\r\\n\\r\\n"
                                    "message...\n");

                    if (!conn->short_preview) {
                        temp = iserver_writeline(conn,
                              "ICAP/1.0 100 Continue\r\n\r\n");
                        if (temp<0)
                              goto done;
                    }

                    if ((conn->icap_reqmod_post ||
                         (conn->icap_req_type == ICAP_RESPMOD))&&
                        conn->short_preview) {

                        iserver_send_modified_response(conn);

                    }

                }

                /* total msg body < preview or 204 sent */
                if (response || conn->short_preview)
                    goto done;

            }

            /* get remaining msg body */
            if ( conn->icap_reqmod_post ||
                 (conn->icap_req_type == ICAP_RESPMOD
                      &&conn->origin_res_body_exist)) {

                if (read_body_chunked(conn) < 0)
                    goto done;
            }

            /* send to api: initial+remaining msg body (if preview >=
0)
             * or total body (if preview <0)
             */
            iserver_send_modified_response(conn);

        done:
            if (CLOSECONN)
              break;
```

```
            if (!conn->closed) {
                  /* keep connection open for next request */
                  iserver_reset_conn(conn);
                  if (cinfo.verbose)
                        iserver_logmsg(0, "Recycling- curr conn active:
%d\n", cinfo.conn_active);

            }
            else {
                  /* close connection */
                  break;
            }

      }

      fflush(0);

#ifdef WINDOWS
      closesocket(conn->skt);
#else
      close(conn->skt);
#endif

      cinfo.conn_active--;

      if (cinfo.verbose)
            iserver_logmsg(0, "Connections active: %d\n",
cinfo.conn_active);

      iserver_free_conn(conn);

}

/* parse the body of the transmission
 */
static int
iserver_parse_body(iserver_conn_t *conn)
{
      char *line;

      assert(conn);

      if (conn->icap_req_type == ICAP_RESPMOD && conn-
>origin_res_hdr_exist) {

            if (!conn->origin_hdr->length) {
                  iserver_logmsg(conn, "TCP connection closed");
                  conn->closed = 1;
                  return -1;
            }

            line = conn->origin_hdr->data;

            if ( strstr(line, "Content-Type:") ) {
                  if ( strstr(line, "Content-Type: text/html") ||
                       strstr(line, "Content-Type: text/text") ) {
                        if (cinfo.verbose)
```

```
                              iserver_logmsg(conn, "HTML content
found\n");
                        conn->html_content = 1;

                  } else {
                        if (cinfo.verbose)
                              iserver_logmsg(conn,
                                          "Unknown content type "
                                          "(probably an image)\n");
                        conn->html_content = 0;
                  }

            } else  {
                  /* no content type specified; assume text/html */
                  conn->html_content = 1;
            }
      }

      return read_preview_chunked(conn);
}

/* write (null-terminated) string str to the connection represented by
conn */
int
iserver_writeline(iserver_conn_t *conn, const char *str)
{
      unsigned int bytes = 0;

      assert(conn && str);

      bytes = WRITE(conn->skt, str, strlen(str));

      if (errno == SIGPIPE) {
            iserver_logmsg(conn, "TCP connection closed");
            conn->closed=1;
            return -1;
      }

      if (bytes < 0) {
            printf("socket write error");
            return bytes;
      }

      assert(bytes == strlen(str));
      return bytes;
}

/* write ca_t to the connection represented by conn */
int
iserver_writereply(iserver_conn_t *conn, const ca_t *str)
{
      int written = 0;

      assert(conn && str);

      /* Sanity checking for burt 42740 */
      assert(strstr(str->beg, "ICAP/1.0") ||
```

```
             strstr(str->beg, "HTTP"));
      /* Sanity checking for burt 46326 */
      assert(strstr(str->beg, "ISTAG"));

      while (written < (str->len-1)) {
            written += WRITE(conn->skt, str->beg, (str->len-1));
            if (errno == EPIPE) {
                  iserver_logmsg(conn, "TCP connection closed\n");
                  conn->closed=1;
                  return -1;
            }
            if (written < 0) {
                  printf("socket write error");
                  return written;
            }
      }

      if (cinfo.verbose) {
            printf("bytes written at iserver_writereply: %d", written);
            printf("Reply written:\n%s", str->beg);
      }
      return written;
}

/*
 * read a line from the connection represented by conn into buf,
 * return num bytes read
 */
int
iserver_readline(iserver_conn_t *conn, char *buf, int buflen)
{
      char c;
      int i, retval;

      assert(conn && buf);

      i = 0;
      for (;;) {

        if ( (retval = READ(conn->skt, &c, 1)) <= 0)
          break;

            buf[i++] = c;
            if (c == '\n')
                  break;

            if (i >= buflen)
                  break;
      }
      if (retval < 0) {
            printf("socket read error");
            return retval;
      }

      if (i >= buflen) {
            printf("\n *** Buffer size exceeded in iserver_readline
***\n");
```

```
            assert(0);
      }
      buf[i] = 0;

      return i;
}

/* allocate an iserver_conn_t, set defaults, and return it */
static iserver_conn_t *
iserver_alloc_conn(void)
{
      iserver_conn_t *c;

      c = (iserver_conn_t *)malloc(sizeof(iserver_conn_t));
      memset(c, '0', sizeof(iserver_conn_t));

      if (!c) {
            printf("malloc error");
            exit(-1);
      }

      c->chunked = 0;
      c->chunk_size = 0;
      c->content_len = -1;
      c->resp_code = 0;
      c->html_content = 0;
      c->closed = 0;
      c->preview = -1;
      c->short_preview = 0;
      c->icap_req_type = -1;
      c->icap_uri = 0;
      c->icap_reqmod_post = 0;
      c->origin_res_hdr_exist = 0;
      c->origin_res_body_exist = 0;

      if (! (c->content = gbuf_alloc(CONTENT_BLK_SIZE)) ) {
            printf("gbuf allocation");
            return 0;
      }
      if (! (c->req_hdr = gbuf_alloc(CONTENT_BLK_SIZE)) ) {
            printf("gbuf allocation");
            return 0;
      }
      if (! (c->client_hdr = gbuf_alloc(CONTENT_BLK_SIZE)) ) {
            printf("gbuf allocation");
            return 0;
      }
      if (! (c->origin_hdr = gbuf_alloc(CONTENT_BLK_SIZE)) ) {
            printf("gbuf allocation");
            return 0;
      }

      return c;
}

/* free an iserver_conn_t */
static void
```

```
iserver_free_conn(iserver_conn_t *conn)
{
      assert (conn);

      if (conn->icap_uri)
              free(conn->icap_uri);

      if (conn->req_hdr)
            gbuf_free(conn->req_hdr);

      if (conn->client_hdr)
            gbuf_free(conn->client_hdr);

      if (conn->origin_hdr)
            gbuf_free(conn->origin_hdr);

      if (conn->content)
            gbuf_free(conn->content);

      free(conn);
}


/* reset an iserver_conn_t
 */
static void
iserver_reset_conn(iserver_conn_t *conn)
{
      assert(conn);

      conn->chunked = 0;
      conn->chunk_size = 0;
      conn->content_len = -1;
      conn->resp_code = 0;
      conn->html_content = 0;
      conn->closed = 0;
      conn->preview = -1;
      conn->short_preview = 0;
      conn->icap_req_type = -1;
      conn->icap_uri = 0;
      conn->icap_reqmod_post = 0;

      if (conn->icap_uri)
              free(conn->icap_uri);
      conn->icap_uri = 0;

      gbuf_zero(conn->req_hdr);
      gbuf_zero(conn->client_hdr);
      gbuf_zero(conn->origin_hdr);
      gbuf_zero(conn->content);

      return;
}

/* print (date and?) timestamped log messages to stderr */
void
iserver_logmsg(iserver_conn_t *conn, const char *fmt,...)
```

```
{
	struct tm *timeptr;
	va_list ap;
	time_t t;

	assert(fmt);

	PTHREAD_MUTEX_LOCK(time_mtx);

	time(&t);
	timeptr = localtime(&t);

	fprintf(stderr, "[");
	if (timeptr->tm_hour < 10)
		fprintf(stderr, "0");
	fprintf(stderr, "%d:", timeptr->tm_hour);

	if (timeptr->tm_min < 10)
		fprintf(stderr, "0");
	fprintf(stderr, "%d:", timeptr->tm_min);

	if (timeptr->tm_sec < 10)
		fprintf(stderr, "0");
	fprintf(stderr, "%d", timeptr->tm_sec);

	  PTHREAD_MUTEX_UNLOCK(time_mtx);

	if (conn)
		fprintf(stderr, " (c%d)", conn->num);

	fprintf(stderr, "] ");

	va_start(ap,fmt);
	vfprintf(stderr, fmt, ap);

	va_end(ap);

	return;
}
```

C. threadpool.c

```
/*
 * $Id: //depot/prod/ontap/main/tools/icap/iserver10/threadpool.c#6 $
 *
 * Copyright (c) 2001 Network Appliance, Inc.
 * All rights reserved.
 */

#include "threadpool.h"
#include "queue.h"
#include <stdio.h>
#include <assert.h>
#ifdef WINDOWS
#include <Windows.h>
#include <errno.h>
#else
#ifdef COMPILE_SOLARIS
#include <errno.h>
#define sem_wait  sema_wait
#define sem_post  sema_post
#define sem_init  sema_init
#else
#include <asm/errno.h>
#endif
#endif

typedef struct pool_thread {
      int         id;
#ifdef WINDOWS
      HANDLE          thr;
      HANDLE      mtx;
      HANDLE      cond;
#else
        pthread_t        thr;
      pthread_mutex_t  mtx;
      pthread_cond_t   cond;
#endif
      void      *arg;

      CIRCLEQ_ENTRY(pool_thread)    q_link;
} pool_thread_t;

typedef CIRCLEQ_HEAD(threadpool_s, pool_thread) threadpool_t;

/*
 * PROTOTYPES
 */
static int threadpool_create_threads(void);
static void threadpool_start_thread(void *arg);
static void threadpool_return_thread(pool_thread_t *thr);

/*
 * GLOBALS
 */
static threadpool_t pool;
```

```
static int num_used;
static void *(*job_func)(void *);
static int num_thr;
#ifdef WINDOWS
HANDLE available_sem;
static HANDLE tp_mtx;          /* for coarse-grained locking */
static HANDLE tp_thrsig_mtx;   /* guarantees threads are ready to be
used */
#else
sem_t available_sem;
static pthread_mutex_t tp_mtx; /* for coarse-grained locking */
static pthread_mutex_t tp_thrsig_mtx; /* guarantees threads are ready
to be used */
#endif

/* Initialize the threads */
int
threadpool_init(int num, void *(*func)(void *))
{
      assert(num >= 0 && func);

      job_func = func;
      num_thr = num;
      num_used = 0;
      /* initially, no threads are available */
#ifdef WINDOWS
      available_sem = CreateSemaphore(NULL, 0, num_thr, NULL);
      tp_mtx = CreateMutex(NULL, FALSE, NULL);
      tp_thrsig_mtx = CreateMutex(NULL, FALSE, NULL);
#else
      sem_init(&available_sem, 0, 0);
      pthread_mutex_init(&tp_mtx, 0);
      pthread_mutex_init(&tp_thrsig_mtx, 0);
#endif
      CIRCLEQ_INIT(&pool);

      if (threadpool_create_threads() < 0) {
            perror("threadpool_create_threads");
            return -1;
      }
      return 0;
}

static int
threadpool_create_threads(void)
{
      int i;
      for (i=0; i < num_thr; i++) {
            pool_thread_t *thr;

            thr = (pool_thread_t*)malloc(sizeof(pool_thread_t));
            if (!thr) {
                  perror("threadpool_create_threads: malloc error");
                  return -1;
            }
            memset(thr, '0', sizeof(pool_thread_t));
```

```
            /* init data fields of thread structure */
            thr->arg = 0;
            thr->id = i+1;
#ifdef WINDOWS
            thr->mtx  = CreateMutex(NULL, FALSE, NULL);
            thr->cond = CreateSemaphore(NULL, 0, 1, NULL);

            if ((thr->thr=CreateThread(NULL,
                                0,

(LPTHREAD_START_ROUTINE)threadpool_start_thread,
                                (void*) thr,
                                0,
                                NULL)) == NULL) {
              perror("thread_create error");
              return -1;
            }
#else
            pthread_mutex_init(&thr->mtx, 0);
            pthread_cond_init(&thr->cond, 0);
            if (pthread_create(&thr->thr,
                           0,
                           (void *(*)(void*))threadpool_start_thread,
                           (void *)thr) != 0) {
                 perror("pthread_create error");
                 return -1;
            }
#endif
          PTHREAD_MUTEX_LOCK(tp_mtx);
          num_used++; /* artificial inflation, but it works */
          PTHREAD_MUTEX_UNLOCK(tp_mtx);
      }
      return 0;
}

static void
threadpool_start_thread(void *arg)
{
      pool_thread_t *thr;

      assert(arg);

      thr = (pool_thread_t*)arg;

      for (;;) {

            PTHREAD_MUTEX_LOCK(tp_thrsig_mtx);
            /* put thread in pool */
            threadpool_return_thread(thr);
#ifdef WINDOWS
            PTHREAD_MUTEX_UNLOCK(tp_thrsig_mtx);
            /* wait to be called into service */
            WaitForSingleObject(thr->cond, INFINITE);
#else
            /* wait to be called into service */
            pthread_cond_wait(&thr->cond, &tp_thrsig_mtx);
            /* MUST BE DONE IMMEDIATELY! */
```

```
            PTHREAD_MUTEX_UNLOCK(tp_thrsig_mtx);
#endif
            /* do work */
            job_func(thr->arg);
      }
}


void
threadpool_dispatch(void *arg)
{
      pool_thread_t *thr;

      assert(arg);

      /* it should be impossible for the list to be empty at this point
*/
      assert( &pool != (threadpool_t*)(pool.cqh_first));

      PTHREAD_MUTEX_LOCK(tp_thrsig_mtx);
      PTHREAD_MUTEX_LOCK(tp_mtx);
      thr = pool.cqh_first;
      CIRCLEQ_REMOVE(&pool, thr, q_link);
      thr->arg = arg;
      num_used++;
      PTHREAD_MUTEX_UNLOCK(tp_mtx);
      PTHREAD_MUTEX_UNLOCK(tp_thrsig_mtx);

      /* wake the thread up to begin its execution */
#ifdef WINDOWS
      ReleaseSemaphore(thr->cond, 1, NULL);
#else
      pthread_cond_signal(&thr->cond);
#endif

}

static void
threadpool_return_thread(pool_thread_t *thr)
{
      assert(thr);

      PTHREAD_MUTEX_LOCK(tp_mtx);
      num_used--;
      CIRCLEQ_INSERT_TAIL(&pool, thr, q_link);
#ifdef WINDOWS
      ReleaseSemaphore(available_sem, 1, NULL);
#else
      sem_post(&available_sem);
#endif
      PTHREAD_MUTEX_UNLOCK(tp_mtx);
}

/* caught Control-C, now call cleanup routine */
void
threadpool_signal_handler(int signum)
{
```

```
	printf("caught control-C\n");
	threadpool_shutdown();
}

/* cleanup the threads, synchronization constructs */
void
threadpool_shutdown(void)
{
	pool_thread_t* thr;
	int i = num_thr;

	if (!i)
		return;

	PTHREAD_MUTEX_LOCK(tp_mtx);
	i -= num_used;
	num_thr=0;

	for (; i>0; i--) {
		thr = pool.cqh_first;
		CIRCLEQ_REMOVE(&pool, thr, q_link);
#ifdef WINDOWS
		CloseHandle(thr->mtx);
		CloseHandle(thr->cond);
#else
		pthread_mutex_destroy(&thr->mtx);
		pthread_cond_destroy(&thr->cond);
#endif
		free(thr);
	}

	PTHREAD_MUTEX_UNLOCK(tp_mtx);
#ifdef WINDOWS
	CloseHandle(tp_mtx);
	CloseHandle(tp_thrsig_mtx);
	CloseHandle(available_sem);
#else
	pthread_mutex_destroy(&tp_mtx);
	pthread_mutex_destroy(&tp_thrsig_mtx);
	sem_destroy(&available_sem);
#endif
	printf("All owned threads destroyed\n");
	if (options_str) {
		free(options_str);
		options_str=0;
	}
	exit(-1);
}
```

D. header.c

```
/*
 * $Id: //depot/prod/ontap/main/tools/icap/iserver10/header.c#8 $
 *
 * Copyright (c) 2001 Network Appliance, Inc.
 * All rights reserved.
 */

#include "header.h"
#include "iserver.h"
#include "gbuf.h"
#include <stdio.h>
#include <assert.h>
#ifdef WINDOWS
#include <errno.h>
#else
#ifdef COMPILE_SOLARIS
#include <errno.h>
#else
#include <sys/errno.h>
#endif
#endif /* Unix/Windows */

/* Parse the ICAP header */
int
header_read_icap(iserver_conn_t *conn)
{
      int retval, icap_req_recognized=0;
      char buf[BUFLEN], *line;

      assert(conn);

      for (;;) {
            memset(buf, '\0', BUFLEN);

            retval = iserver_readline(conn,buf,BUFLEN);
            if (retval < 0) {
                  if (errno == EPIPE) { /* 32 */
                        iserver_logmsg(conn, "TCP connection
closed\n");
                  }
                  conn->closed = 1;
                  goto header_error;

            } else if (retval == 0)
                  break;

            line = buf;

            if (line) {

                  if (!MATCH(line, CRLF)) {

                        if (cinfo.verbose)
                              printf("Server-rhdr : %s", line);
```

```
                        if (strstr(line, "res-hdr")) {
                              conn->origin_res_hdr_exist = 1;
                        }
                        if (strstr(line, "res-body")) {
                              conn->origin_res_body_exist = 1;
                        }
                        if (MATCH(line, "OPTIONS")) {

                              conn->icap_req_type = ICAP_OPTIONS;
                              icap_req_recognized=1;

                        } else if (MATCH(line, "REQMOD")) {

                              conn->icap_req_type = ICAP_REQMOD;
                              icap_req_recognized=1;

                        } else if (MATCH(line, "RESPMOD")) {

                              conn->icap_req_type = ICAP_RESPMOD;
                              icap_req_recognized=1;

                        } else if (MATCH(line, "LOG")) {

                              conn->icap_req_type = ICAP_LOG;
                              icap_req_recognized=1;

                        } else      if (MATCH(line, "Transfer-Encoding:
chunked")) {

                              iserver_logmsg(conn, "Transfer-Encoding
field cannot exist for ICAP 1.0\n");
                              goto header_error;

                        } else if (MATCH(line, "Preview:")) {
                              if (!sscanf(line, "Preview:%d", &conn-
>preview))
                                    conn->preview = 0;
                              line = 0;
                        } else if (MATCH(line, "TE:")) {
                              /* NO-OP */
                        } else {
                              /* Dont want no headers besides
                                 one's we recognize */
                              line = 0;
                        }
                  }

                  /* add to request header */
                  gbuf_concat(conn->req_hdr, line, retval);

                  if ( line && MATCH(line, CRLF)) {
                        if (cinfo.verbose)
                              printf("Server-RXXX : %s", line);
                        break;
                  }
            }
      }
```

```
      if (icap_req_recognized)
            return 1;
      else {
            if (!retval)
                  iserver_logmsg(conn, "conn closed by client\n");
            else
                  iserver_logmsg(conn, "unrecognized ICAP request\n");

            conn->closed=1;
            return -1;
      }

  header_error:
      iserver_logmsg(conn, "error while parsing ICAP header\n");
      return -1;
}

/* parse HTTP client header */
int
header_read_client(iserver_conn_t *conn)
{
      char buf[BUFLEN];
      int retval;

      for (;;) {
            char *line;

            memset(buf, '\0', BUFLEN);
            retval = iserver_readline(conn,buf,BUFLEN);

            if (retval < 0) {
                  if (errno == EPIPE) { /* 32 */
                        iserver_logmsg(conn, "TCP connection
closed\n");
                  }
                  conn->closed = 1;
                  goto header_error;

            } else if (retval == 0)
                  break;

            line = buf;

            if (line) {
                  if (MATCH(line, "POST") &&
                      (conn->icap_req_type == ICAP_REQMOD)) {
                        conn->icap_reqmod_post = 1;
                  }

                  gbuf_concat(conn->client_hdr, line, retval);

                  if (cinfo.verbose)
                        printf("Server-chdr : %s", line);

                  if (MATCH(line, CRLF))
                        break;
```

```
            }
      }
      return 1;

  header_error:
      iserver_logmsg(conn,"error while parsing HTTP client request
hdr\n");
      return -1;
}

/* parse HTTP origin header for respmod */
int
header_read_origin(iserver_conn_t *conn)
{
      char buf[BUFLEN];
      int retval;

      for (;;) {
            char *line;

            memset(buf, '\0', BUFLEN);
            retval = iserver_readline(conn,buf,BUFLEN);
            if (retval < 0) {
                  if (errno == EPIPE) { /* 32 */
                        iserver_logmsg(conn,"TCP connection closed\n");
                  }
                  conn->closed = 1;
                  goto header_error;

            } else if (retval == 0)
                  break;

            line = buf;

            if (line) {
                  if (!MATCH(line, CRLF)) {
                        if (cinfo.verbose)
                              printf("Server-ohdr : %s", line);

                  }
                  if (MATCH(line, "HTTP/1.1")) {
                        sscanf(line,
                               "HTTP/1.1 %d",
                               &conn->resp_code);
                  }
                  if (MATCH(line, "Via")) {
                        /* insert the hostname in Via Field */
                        if (*(line+strlen(line)-1) == '\n') {
                              *(line+strlen(line)-1) = '\0';
                              if (*(line+strlen(line)-1) == '\r') {
                                    *(line+strlen(line)-1) = '\0';
                              }
                        }
                        memcpy(line+strlen(line), ", ", 2);
                        assert(*hostname);
                        memcpy(line+strlen(line), hostname,
                               strlen(hostname));
```

```
                        memcpy(line+strlen(line), "\r\n", 2);
                        retval=strlen(line);
                }

                gbuf_concat(conn->origin_hdr, line, retval);

                if (line && MATCH(line, CRLF)) {
                        if (cinfo.verbose)
                                printf("Server-OXXX : %s", line);
                        break;
                }
        }
    }
    return 1;

 header_error:
    iserver_logmsg(conn, "error while parsing origin header\n");
    return -1;
}
```

E. stream.c

```
/*
 * $Id:
 *
 * Copyright (c) 2001 Network Appliance, Inc.
 * All rights reserved.
 */

#include "stream.h"
#include "iserver.h"
#include "gbuf.h"
#include <assert.h>
#include <stdio.h>

#define STREAM_READ_FAILED -1

int
read_preview_chunked(iserver_conn_t *conn)
{
      int nread, bytes, read_len;
      char *p;
      char buf[BUFLEN];
      char *dbuf = 0;

      assert(conn);

      nread = 0;
      while (nread <= conn->preview) {

            memset(buf, '\0', BUFLEN);
            if ( (bytes = iserver_readline(conn, buf, BUFLEN)) <= 0 ) {
                  iserver_logmsg(conn,
                              "error reading chunked\n");
                  nread = STREAM_READ_FAILED;
                  conn->closed = 1;
                  goto done;
            }

            /* chop carriage-return linefeed */
            assert( p = (char *)strstr(buf, CRLF) );
            if (p) {
                  *p = 0;
                  *(p+1) = 0;
            }

            /* convert hex chunk size to decimal */
            if (!sscanf(buf, "%x", &conn->chunk_size)) {
                  iserver_logmsg(conn, "error converting chunk
size\n");
                  nread = STREAM_READ_FAILED;
                  goto done;
            }

            if (conn->chunk_size == 0) {
                    /* last chunk indicator for preview*/
                  /* check for "0; ieof\r\n"
```

```
                 * to decide if this is end of the body also
                 */
                if (!strcmp(buf, "0; ieof"))
                      conn->short_preview = 1;
                /* read extra \r\n as per correction to ICAP10 spec
*/
                iserver_readline(conn, buf, BUFLEN);
                goto done;
          }

          read_len = (conn->chunk_size > conn->preview) ?
                conn->preview : conn->chunk_size;

          dbuf = (char *)malloc(read_len);
          if (!dbuf) {
                iserver_logmsg(conn, "error allocating dbuf\n");
                goto done;
          }
          memset(dbuf, '\0', read_len);

          while (bytes = READ(conn->skt, dbuf, read_len)) {

                gbuf_concat(conn->content, dbuf, bytes);
                nread += bytes;
                read_len -= bytes;

                if (conn->html_content) {
                      if (cinfo.verbose)
                            printf("\nCTNT (Size %d): %s\n",
                                  conn->content->length,
                                  conn->content->data);
                }

                if (!read_len) {
                      break;
                }

                memset(dbuf, '\0', read_len);

                if (bytes == 0) {
                      iserver_logmsg(conn, "chunk2:  TCP closed!\n");
                      conn->closed = 1;
                      nread = STREAM_READ_FAILED;
                      goto done;
                }
          }

          free(dbuf);
          dbuf = 0;

          if (nread == conn->chunk_size) {
                /* eat trailing \r\n  */
                iserver_readline(conn, buf, BUFLEN);

          }
     }
```

```
      if (cinfo.verbose)
            iserver_logmsg(conn, "Preview size: %d, read: %d", conn-
>preview, nread);

 done:
      if (dbuf)
            free(dbuf);
      return nread;
}

int
read_body_chunked(iserver_conn_t *conn)
{
      int nread = 0;
      int temp = 0;
      int rsize, bytes;
      char *p=0, *dbuf=0, buf[BUFLEN];

      if (conn->preview > 0) {

            assert(conn);

            if (conn->content->length < conn->chunk_size ) {
                  conn->chunk_size -= conn->content->length;
                  goto chunk_partial;
            }
      }

      for (;;) {
            /* eat chunk size*/
            iserver_readline(conn, buf, BUFLEN);
            if (!buf[0]) {
                        /* TCP closed */
                  iserver_logmsg(conn, "blank line close
(surprise)\n");
                  conn->closed = 1;
                  nread = STREAM_READ_FAILED;
                  goto done;
            }

            /* chop carriage-return linefeed */
            assert( p = (char *)strstr(buf, CRLF) );
            if (p) {
                  *p = 0;
                  *(p+1) = 0;
            }

            /* convert hex chunk size to decimal */
            if (!sscanf(buf, "%x", &conn->chunk_size)) {
                  iserver_logmsg(conn, "error converting chunk
size\n");
                  nread = STREAM_READ_FAILED;
                  goto done;
            }

            if (conn->chunk_size == 0) {
                  /* last chunk indicator for msg body */
```

```
                  /* received "0; ieof\r\n"  or "0\r\n" */

                  /* read extra \r\n as per correction to ICAP10 spec
*/
                  iserver_readline(conn, buf, BUFLEN);
                  goto done;
            }

            /* eat chunk body */
      chunk_partial:
            rsize = conn->chunk_size;

            dbuf = (char *)malloc(conn->chunk_size);
            if (!dbuf) {
                  perror("malloc error");
                  nread = STREAM_READ_FAILED;
                  goto done;
            }
            memset(dbuf, '\0', conn->chunk_size);

            while (bytes = READ(conn->skt, dbuf, rsize)) {

                  nread += bytes;

                  if (bytes == 0) {
                              /* TCP closed */
                        if (cinfo.verbose)
                              iserver_logmsg(conn, "chunk2: TCP closed
(partial chunk)!\n");
                        conn->closed = 1;
                        nread = STREAM_READ_FAILED;
                        goto done;
                  }

                  if(conn->html_content) {
                        if (cinfo.verbose)
                              printf("CTNT:  %s\n",dbuf);
                  }

                  gbuf_concat(conn->content, dbuf, bytes);

                  rsize-= bytes;

                  if (!rsize){
                        /* eat trailing \n */
                        iserver_readline(conn, buf, BUFLEN);
                        break;
                  }

                  memset(dbuf, '\0', conn->chunk_size);
            }

            free(dbuf);
            dbuf=0;
      }
 done:
      if (dbuf)
```

```
            free(dbuf);
        return nread;
}
```

F. api.c

```
/*
 * $Id: //depot/prod/ontap/main/tools/icap/iserver10/api.c#26 $
 *
 * Copyright (c) 2001 Network Appliance, Inc.
 * All rights reserved.
 */

#include "api.h"
#include "api_filter.h"
#ifndef COMPILE_SOLARIS
#include "banner_stripper.h"
#endif

/* GLOBALS */
int modify_control=0;
int cache_control=0;

char*
api_options_response(const char* icap_uri)
{
#ifdef ECHO_SERVER
      /* return a pointer to the options response string */
      FILE *fp;
      char options[3000];

      memset(options, '\0', 3000);
      assert(options_str);
      memcpy(options, options_str, strlen(options_str));
      memcpy(options+strlen(options), "\r\n", 2);

      if (cinfo.verbose)
            printf("api:Sending options:\n%s", options);
      return (char*) strdup(options);
#endif
}

void
api_log_request(const char* icap_hdr)
{
      /* NO-OP */
}

char*
api_preview_reqmod(const char* icap_hdr,
               const char* client_hdr,
               const char* post_msg_body,
               int post_msg_body_len,
               int preview_size)
{
#ifdef ECHO_SERVER
      char *scratch, *line;
      char str[300], *trans_id_str, *trans_id_str_end;

      char not_interested_header[] = "ICAP/1.0 204 No Content\r\n";
      int msg_size=0, trans_id_len=0, offset;
```

```
      char* not_interested_msg;

      assert(icap_hdr&&client_hdr&&post_msg_body);

      if (cinfo.modify_percent)
            return NULL;
      else {
            /* Construct 204 with IS-TAG header and Transaction ID
               (if it exists) */
              memset(str, '\0', 300);
            assert(options_str);
            assert(line=(char*)strstr(options_str, "ISTAG"));
            while (*line != '\n')
                  memcpy(str+strlen(str), line++, 1);

            msg_size = strlen(not_interested_header);
            msg_size += strlen(str);

            trans_id_str = (char*) strstr(icap_hdr, "Transaction-Id:");

            if (trans_id_str) {
                     trans_id_str_end = (char*) strstr(trans_id_str,
"\n");
                   scratch = trans_id_str;
                   trans_id_str_end++;
                   while(scratch != trans_id_str_end) {
                           trans_id_len++;
                         scratch++;
                   }
                   msg_size += trans_id_len;
            }

            not_interested_msg = (char*)
malloc((msg_size+5)*sizeof(char));
            memset(not_interested_msg, '\0',
(msg_size+5)*sizeof(char));
            memcpy(not_interested_msg,
                  not_interested_header,
                  strlen(not_interested_header));
            offset = strlen(not_interested_header);
            memcpy(not_interested_msg+offset, str, strlen(str));
            offset += strlen(str);
            memcpy(not_interested_msg+offset, "\r\n", 2);
            offset += 2;
            if (trans_id_len) {
                  memcpy(not_interested_msg+offset,
                         trans_id_str,
                         trans_id_len);
                  offset += trans_id_len;
            }
            memcpy(not_interested_msg+offset, "\r\n", 2);
            return not_interested_msg;
      }
#endif
}

static int
```

```
api_respmod_200_or_204(const char* origin_resp_hdr, int
origin_resp_body_len)
{
      int modify_me = 0;
#ifdef BANNER_STRIPPER
      modify_me = 1;
#endif
      /* For Valley Girl filtering - we can modify content-type
         advertised as html or text only */
      if (*origin_resp_hdr && strstr(origin_resp_hdr, "Content-Type:"))
            {
                  if (strstr(origin_resp_hdr,"Content-Type: text"))
                        modify_me = 1;
            }
      else
            modify_me = 1;

      modify_control++;
      if (modify_me && origin_resp_body_len &&
          ((modify_control%100) <= cinfo.modify_percent))
            return 1;
      else
            return 0;
}

char*
api_preview_respmod(const char* icap_hdr,
                const char* client_hdr,
                const char* origin_resp_hdr,
                const char* origin_resp_body,
                int origin_resp_body_len,
                int preview_size)
{
#ifdef ECHO_SERVER
      FILE *fp;
      char *line, *scratch;
      char str[300], *trans_id_str, *trans_id_str_end;

      char not_interested_header[] = "ICAP/1.0 204 No Content\r\n";
      int msg_size=0, trans_id_len=0, offset;
      char* not_interested_msg;

      assert(icap_hdr&&client_hdr&&origin_resp_hdr&&origin_resp_body);

      if (api_respmod_200_or_204(origin_resp_hdr,
origin_resp_body_len))
            return NULL;
      else {
            /* Construct 204 with IS-TAG header and Transaction ID
               (if it exists) */
            memset(str, '\0', 300);
            assert(options_str);
            assert(line=(char*)strstr(options_str, "ISTAG"));
            while (*line != '\n')
                  memcpy(str+strlen(str), line++, 1);

            msg_size = strlen(not_interested_header);
```

```
            msg_size += strlen(str);

            trans_id_str = (char*) strstr(icap_hdr, "Transaction-Id:");

            if (trans_id_str) {
                     trans_id_str_end = (char*) strstr(trans_id_str,
"\n");
                   scratch = trans_id_str;
                   trans_id_str_end++;
                   while(scratch != trans_id_str_end) {
                           trans_id_len++;
                         scratch++;
                   }
                   msg_size += trans_id_len;
            }

            not_interested_msg = (char*)
malloc((msg_size+5)*sizeof(char));
            memset(not_interested_msg, '\0',
(msg_size+5)*sizeof(char));
            memcpy(not_interested_msg,
                  not_interested_header,
                  strlen(not_interested_header));
            offset = strlen(not_interested_header);
            memcpy(not_interested_msg+offset, str, strlen(str));
            offset += strlen(str);
            memcpy(not_interested_msg+offset, "\r\n", 2);
            offset += 2;
            if (trans_id_len) {
                  memcpy(not_interested_msg+offset,
                        trans_id_str,
                        trans_id_len);
                  offset += trans_id_len;
            }
            memcpy(not_interested_msg+offset, "\r\n", 2);
            return not_interested_msg;
      }
#endif
}

static int
api_reqmod_cachable()
{
      if (((cache_control++)%100)>=cinfo.reqmod_cache_percent)
            return 0;
      return 1;
}

ca_t*
api_modify_reqmod(const char* icap_hdr,
              const char* client_hdr,
              const char* post_msg_body,
              int post_msg_body_len)
{
#ifdef ECHO_SERVER
      /* reconstruct message to echo back to client*/
      int i, msg_size[7] = {0,0,0,0,0,0,0};
```

```
char *result, scratch[200], str[600], body_size_str[20], *line;
FILE* fp;
ca_t* res = (ca_t*) malloc(sizeof(ca_t));

res->beg = 0;
res->len = 0;

assert(icap_hdr && client_hdr && post_msg_body);

if (cinfo.verbose) {
      printf("api: icap_hdr METHOD:\n%s", icap_hdr);
      printf("api: client_hdr:\n%s", client_hdr);
      printf("api: post_msg_body:\n%s", post_msg_body);
}

/* read out the 200 OK line */
memset(str, '\0', 300);
assert(options_str);
assert(line=(char*)strstr(options_str, "ICAP"));
while (*line != '\n')
      memcpy(str+strlen(str), line++, 1);
memcpy(str+strlen(str), "\r\n", 2);

/* XXX this MUST return the IS-TAG line */
assert(options_str);
assert(line=(char*)strstr(options_str, "ISTAG"));
while (*line != '\n')
      memcpy(str+strlen(str), line++, 1);
memcpy(str+strlen(str), "\r\n", 2);
assert(strstr(str, "ISTAG"));

/* Generate Cache-Control and Attribute header */
memset(scratch, '\0', 100);
if (api_reqmod_cachable()) {
      memcpy(scratch, "Cache-Control: max-age=360000\r\n",
             strlen("Cache-Control: max-age=360000\r\n"));
      memcpy(scratch+strlen(scratch), "Attribute: none\r\n",
             strlen("Attribute: none\r\n"));
}

/* Generate Encapsulated header */
if (post_msg_body_len) {
      memcpy(scratch+strlen(scratch),
             "Encapsulated: req-hdr=0, req-body=",
             strlen("Encapsulated: req-hdr=0, req-body="));
      sprintf(scratch+strlen(scratch), "%d", strlen(client_hdr));
      memcpy(scratch+strlen(scratch), "\r\n\r\n", 4);
}
else {
      memcpy(scratch+strlen(scratch),
             "Encapsulated: req-hdr=0\r\n\r\n",
             strlen("Encapsulated: req-hdr=0\r\n\r\n"));
}

/* ICAP REQMOD response hdr: 2OO OK + ISTAG + Encapsulated +
   Cache-Control*/
msg_size[0] = strlen(str);
```

```
      msg_size[1] = strlen(scratch);
        msg_size[2] = strlen(client_hdr);
      if (post_msg_body_len) {
            msg_size[4] = post_msg_body_len;
            sprintf(body_size_str, "%x", msg_size[4]);
            msg_size[3] = strlen(body_size_str)+2;
      }

      for(i=0; i<5; i++)
            msg_size[5] += msg_size[i];

      if (post_msg_body_len) {
            /* 8 = 2 "\r\n" terminate post body
                 + 5 "0\r\n\r\n" Last chunk
                 + 1 null term. */
            msg_size[6] = (msg_size[5]+8)*sizeof(char);
      }
      else {
            /* 1 = null term. */
            msg_size[6] = (msg_size[5]+1)*sizeof(char);
      }
      result = (char*) malloc(msg_size[6]);
      memset(result, '\0', msg_size[6]);

      memcpy(result, str, msg_size[0]);
      memcpy(result+strlen(result), scratch, msg_size[1]);
      memcpy(result+strlen(result), client_hdr, msg_size[2]);
      if (post_msg_body_len) {
            line = result+strlen(result);
            memcpy(line, body_size_str, msg_size[3]);
            line += msg_size[3]-2;
            memcpy(line, "\r\n", 2);
            line += 2;
            memcpy(line, post_msg_body, msg_size[4]);
            line += msg_size[4];
            memcpy(line, "\r\n0\r\n\r\n", 7);
      }

      if (cinfo.verbose)
            printf("api: icap modified post body:\n%s", result);
      res->beg = result;
      res->len = msg_size[6];
      return res;
#endif
}

ca_t*
api_modify_respmod(const char* icap_hdr,
               const char* client_hdr,
               const char* origin_resp_hdr,
               const char* origin_resp_body,
               int origin_resp_body_len)
{
#ifdef ECHO_SERVER
      /* reconstruct message to echo back to client*/
      int msg_size[7] = {0,0,0,0,0,0,0};
      int i, modify_me=0, illegal_banner=0, trailers_needed=0;
```

```
      char *result, body_size_str[20], *line, str[600];
      char trailers[] = "Cache-Control: no-cache\r\nConnection:
close\r\n";
      FILE *fp;
      gbuf_t* new_body;
      ca_t* res = (ca_t*) malloc(sizeof(ca_t));
      res->beg = 0;
      res->len = 0;

      assert(icap_hdr && client_hdr && origin_resp_hdr &&
origin_resp_body);

      memset(str, '\0', 600);
      if (cinfo.verbose) {
            printf("api: icap_hdr METHOD:\n%s", icap_hdr);
            printf("api: client_hdr:\n%s", client_hdr);
            printf("api: origin_resp_hdr:\n%s", origin_resp_hdr);
      }

      /* read out the 200 OK line */
      memset(str, '\0', 300);
      assert(options_str);
      assert(line=(char*)strstr(options_str, "ICAP"));
      while (*line != '\n')
            memcpy(str+strlen(str), line++, 1);
      memcpy(str+strlen(str), "\r\n", 2);

      /* XXX this MUST return the IS-TAG line */
      assert(options_str);
      assert(line=(char*)strstr(options_str, "ISTAG"));
      while (*line != '\n')
            memcpy(str+strlen(str), line++, 1);
      memcpy(str+strlen(str), "\r\n", 2);
      assert(strstr(str, "ISTAG"));

      /* Insert Encapsulated header */
      if (*origin_resp_hdr) {
            memcpy(str+strlen(str), "Encapsulated: res-hdr=0, res-
body=",
                  strlen("Encapsulated: res-hdr=0, res-body="));
            sprintf(str+strlen(str), "%d", strlen(origin_resp_hdr));
            memcpy(str+strlen(str), "\r\n\r\n", 4);
      }
      else if (*origin_resp_body) {
              memcpy(str+strlen(str), "Encapsulated: res-body=0",
                   strlen("Encapsulated: res-body=0"));
            memcpy(str+strlen(str), "\r\n\r\n", 4);
      }
      else
            memcpy(str+strlen(str), "\r\n", 2);

      msg_size[0] = strlen(str);
      msg_size[2] = strlen(origin_resp_hdr);

#ifndef WINDOWS
#ifdef FILTER
      if (strstr(origin_resp_hdr, "Content-Type:")&&
```

```
            origin_resp_body_len)
            {
                  if (strstr(origin_resp_hdr,"Content-Type:
text/html")||
                      strstr(origin_resp_hdr,"Content-Type:
text/text")){
                        modify_me = 1;
                        new_body = api_filter(origin_resp_body,
                                        origin_resp_body_len);
                        origin_resp_body_len = new_body->length;
                        origin_resp_body = new_body->data;
                        free(new_body);
                  }
            }
#endif

#ifdef BANNER_STRIPPER
      if (origin_resp_body_len &&
          *origin_resp_body == 'G' &&
          *(origin_resp_body+1) == 'I' &&
          *(origin_resp_body+2) == 'F') {

            new_body = strip_banner(origin_resp_body,
                              origin_resp_body_len);

            if (new_body->length) {
                  illegal_banner=1;
                  origin_resp_body_len = new_body->length;
                  origin_resp_body = new_body->data;
            }
            else
                    free(new_body->data);
            free(new_body);
      }
#endif
#endif
      /* handle case in which body size=0 */
      msg_size[4] = origin_resp_body_len;
      sprintf(body_size_str, "%x", msg_size[4]);
      msg_size[3] = strlen(body_size_str);

      for(i=0; i<5; i++)
            msg_size[5] += msg_size[i];

      /* determine if trailers need to be added */
      if (cinfo.trailer_support) {
            if (strstr(icap_hdr, "TE")) {
                  trailers_needed = 1;
            }
      }

        /* handle case in which body size=0 */
      if (origin_resp_body_len) {
            /* 10 = 2 (\r\n for body_size_str term.)
                  + 2 (\r\n for terminating actual last data chunk)
                  + 5 (0\r\n\r\n for last 0 byte chunk)
                  + 1 (NULL term. for string)
```

```
            */
            msg_size[6] = (msg_size[5]+10)*sizeof(char) +
                  trailers_needed*strlen(trailers);
      }
      else {
            /* 5 = 2 (\r\n for body_size_str term.)
                 + 2 (\r\n for last 0 byte chunk)
                 + 1 (NULL term. for string */
            msg_size[6] = (msg_size[5]+5)*sizeof(char) +
                  trailers_needed*strlen(trailers);
      }
      result = (char*) malloc(msg_size[6]);
      memset(result, '\0', msg_size[6]);

      memcpy(result, str, msg_size[0]);
      i = msg_size[0];

      if (*origin_resp_hdr) {
            memcpy(result+i, origin_resp_hdr, msg_size[2]);
            i += msg_size[2];
      }

      memcpy(result+i, body_size_str, msg_size[3]);
      i += msg_size[3];
      memcpy(result+i, "\r\n", 2);
      i += 2;

      /* handle case in which body size=0 */
      if (origin_resp_body_len) {
            memcpy(result+i, origin_resp_body, msg_size[4]);
            i += msg_size[4];
            if (trailers_needed) {
                  memcpy(result+i, "\r\n0\r\n", 5);
                  i+= 5;
                  memcpy(result+i, trailers, strlen(trailers));
                  i += strlen(trailers);
                  memcpy(result+i, "\r\n", 2);
                  i+= 2;
            }
            else
                  memcpy(result+i, "\r\n0\r\n\r\n", 7);
      }
      else {
            if (trailers_needed) {
                  memcpy(result+i, trailers, strlen(trailers));
                  i += strlen(trailers);
                  memcpy(result+i, "\r\n", 2);
                  i+= 2;
            }
            else
                  memcpy(result+i, "\r\n", 2);
      }

#ifndef WINDOWS
#ifdef FILTER
      if (modify_me)
            free(origin_resp_body);
```

```
#endif
#ifdef BANNER_STRIPPER
      if (illegal_banner)
            free(origin_resp_body);
#endif
#endif /* WINDOWS */

      res->beg = result;
      res->len = msg_size[6];
      return res;
#endif /* ECHO_SERVER */

}
```

G. api_filter.c

```
/*
 * $Id:
 *
 * Copyright (c) 2001 Network Appliance, Inc.
 * All rights reserved.
 */

#include "api_filter.h"

gbuf_t*
api_filter(const char* test, int test_len)
{
      int childpid, pipe1[2], pipe2[2];

      if (pipe(pipe1)<0 || pipe(pipe2)<0) {
            printf("Cant create pipe(s)\n");
            exit(-1);
      }

      if ((childpid = fork()) < 0) {
            printf("Cant fork child\n");
            exit(-1);
      } else if (childpid>0) {
            /* parent process */
              gbuf_t *gbresult;
              fd_set  fd_read;
            fd_set  fd_write;
            int bytes_written=0, bytes_read=0, temp, max, closed=0;

            close(pipe1[0]);
            close(pipe2[1]);
            gbresult = gbuf_alloc(CONTENT_BLK_SIZE);
            max = pipe1[1] > pipe2[0] ? pipe1[1] : pipe2[0];
            max++;

            while(1) {
                  FD_ZERO(&fd_read);
                  FD_ZERO(&fd_write);
                  FD_SET(pipe2[0], &fd_read);
                  FD_SET(pipe1[1], &fd_write);

                  if (closed)
                        temp = select(max,
                                    &fd_read,
                                    (fd_set*) 0,
                                    (fd_set*) 0,
                                    (struct timeval*) 0);
                  else
                        temp = select(max,
                                    &fd_read,
                                    &fd_write,
                                    (fd_set*) 0,
                                    (struct timeval*) 0);
```

```
            if (temp < 0) {
                  printf("Select call failed\n");
                  exit(-1);
            }

            if (!closed&&FD_ISSET(pipe1[1], &fd_write)) {
                  int num;
                  num = write(pipe1[1], test++, 1);
                  if (num < 0) {
                        printf("write on pipe failed\n");
                        exit(-1);
                  }
                  else if (!num || bytes_written == test_len) {
                        assert (bytes_written == test_len);
                        close(pipe1[1]);
                        closed=1;
                  }
                  else {
                        bytes_written += num;
                  }
            }

            if (FD_ISSET(pipe2[0], &fd_read)) {
                  int num;
                  char c;

                  num = read(pipe2[0], &c, 1);
                  if (num < 0) {
                        printf("read on pipe failed\n");
                        exit(-1);
                  }
                  else if (!num) {
                        close(pipe2[0]);
                        break;
                  }
                  else {
                        gbuf_concat(gbresult, &c, 1);
                  }
            }
      }

      while (wait((int*) 0) != childpid)
            ;

      /* if (cinfo.verbose)
         printf("api_filter result:\n%s", gbresult->data); */
      return gbresult;
} else {
      /* child process */
      int fd1, fd2;
      char* v_argv[2];

      v_argv[0] = "./valspeak";
      v_argv[1] = NULL;
      close(pipe1[1]);
      close(pipe2[0]);
      dup2(pipe1[0], 0);
```

```
            dup2(pipe2[1], 1);
            /* exec filter here */
            //printf("This page has been \"Valley Girl filtered\" by
ICAP 1.0\n");
            fd1 = execv((char*) v_argv[0], v_argv);
            printf("Error occured with execv - error code: %d\n", fd1);
            close(pipe1[0]);
            close(pipe2[1]);
            exit(0);
      }
}
```

H. valspeak.l

```
%x JAVA STYLE IN_TAG
%%
"<script"               { BEGIN(JAVA);    return(yytext); }
"<SCRIPT"               { BEGIN(JAVA);    return(yytext); }
<JAVA>"</script>"       { BEGIN(0);       return(yytext); }
<JAVA>"</SCRIPT>"       { BEGIN(0);       return(yytext); }
"<style"                { BEGIN(STYLE);   return(yytext); }
"<STYLE"                { BEGIN(STYLE);   return(yytext); }
<STYLE>"</style>"       { BEGIN(0);       return(yytext); }
<STYLE>"</STYLE>"       { BEGIN(0);       return(yytext); }
"<"                     { BEGIN(IN_TAG);  return(yytext); }
<IN_TAG>">"             { BEGIN(0); return(yytext); }
<JAVA>.                 return(yytext);
<STYLE>.                return(yytext);
<IN_TAG>.               return(yytext);
" bad"                  return(" mean");
" big "                 return(" bitchin' ");
" biggest"          return(" bitchin'est");
" body"                 return(" bod");
" bore"                 return(" drag");
" car "                 return(" rod ");
" dirty"            return(" grodie");
" dog"                  return(" dawg");
" everywhere"           return(" all over");
" filthy"           return(" grodie to thuh max");
" food"                 return(" munchies");
" girl"                 return(" chick");
" good"                 return(" bitchin'");
" great"            return(" awesum");
" gross"            return(" grodie");
" guy"                  return(" dude");
" her "                 return(" that chick ");
" her."                 return(" that chick.");
" him "                 return(" that dude ");
" him."                 return(" that dude.");
" can be "          |
" can't be "            |
" should have been "    |
" shouldn't have been " |
" should be "           |
" shouldn't be "    |
" was "                 |
" wasn't "          |
" will be "         |
" won't be "            |
" is "              {
                    ECHO;
                    switch(rand() % 6)
                    {
                    case 0:
                          return(" like, ya know, "); break;
                    case 1:
                          return(" "); break;
                    case 2:
```

```
                        return(" like wow! "); break;
                    case 3:
                        return(" ya know, like, "); break;
                    case 4:
                        return(" "); break;
                    case 5:
                        return(" "); break;
                    }
            }
" house"            return(" pad");
" interesting"          return(" cool");
" large"            return(" awesum");
" leave"            return(" blow");
" man "                 return(" nerd ");
" woman"                return(" gurl");
" maybe "   {
                    switch(rand() % 6)
                    {
                    case 0:
                        return(" if you're a Pisces "); break;
                    case 1:
                        return(" if the moon is full "); break;
                    case 2:
                        return(" if the vibes are right "); break;
                    case 3:
                        return(" when you get the feeling "); break;
                    case 4:
                        return(" maybe "); break;
                    case 5:
                        return(" maybe "); break;
                    }
            }
" meeting"          return(" party");
" movie"            return(" flick");
" music "           return(" tunes ");
" neat"                 return(" keen");
" nice"                 return(" class");
" no way"           return(" just no way");
" people"           return(" guys");
" really"           return(" totally");
" strange"          return(" freaky");
" the "             return(" thuh ");
" very"                 return(" super");
" want"                 return(" wanna");
" weird"            return(" far out");
" yes"                  return(" fer shure");
"But "                  return("Man, ");
"He "               return("That dude ");
"I like"            return("I can dig");
"No,"               return("Like, no way,");
Sir                 return("Man");
"She "                  return("That fox ");
This                return("Like, ya know, this");
There               return("Like, there");
"We "               return("Us guys ");
"Yes,"                  return("Like,");
", "        {
```

```
                    switch(rand() % 6)
                    {
                    case 0:
                          return(", like, "); break;
                    case 1:
                          return(", fer shure, "); break;
                    case 2:
                          return(", like, wow, "); break;
                    case 3:
                          return(", oh, baby, "); break;
                    case 4:
                          return(", man, "); break;
                    case 5:
                          return(", mostly, "); break;
                    }
              }
!             {
                    switch(rand() % 3)
                    {
                    case 0:
                          return("!  Gag me with a SPOOOOON!"); break;
                    case 1:
                          return("!  Gag me with a pitchfork!"); break;
                    case 2:
                          return("!  Oh, wow!");
                    }
              }

ing                 return("in'");
.                   return yytext;
\n                        return("\n");
%%

main()
{
        char* line;
      srand(getpid());
      while(line=yylex())
         write(1, line, strlen(line));
}
```

I. banner_stripper.c

```
/*
 * $Id:
//depot/prod/ontap/main/tools/icap/iserver10/banner_stripper.c#6 $
 *
 * Copyright (c) 2001 Network Appliance, Inc.
 * All rights reserved.
 */

#include "banner_stripper.h"
#include <stdio.h>
#include <assert.h>

/* source http://www.iab.net/iab_banner_standards/bannersizes.html */
image_size_t legal_dimensions[NUM_LEGAL_BANNER] = {
  {60, 468}, {60, 234}, {31, 88}, {90, 120}, {60, 120}, {600, 160},
  {600, 120}, {125, 125}, {150, 180}, {240, 120}, {250, 300},
  {250, 250}, {400, 240}, {280, 336}
};

gbuf_t*
strip_banner(const char* gif_buf, int buf_len){
      int i, valid=0;
      FILE *fp;
      char temp_file[100];
      char temp_buf[2048], netapp_gif[] = "netapp.gif";
      GifFileType *file_type;

      gbuf_t *gbresult = gbuf_alloc(CONTENT_BLK_SIZE);
      assert(gif_buf);
      assert(buf_len);
      /*
        strncpy(temp_file, "/tmp/");
        sprintf(temp_file+5, "%d", rand());
      */
      sprintf(temp_file, "%d", rand());

      /* create a file for the buf */
      if ((fp = fopen(temp_file, "wb")) == NULL) {
            printf("Cannot create temp file for gif\n");
            exit(-1);
      }
      for(i=0; i<buf_len; i++) {
            if (putc(gif_buf[i], fp) == EOF) {
                  printf("Error writing to temp file\n");
                  exit(-1);
            }
      }
      if (fclose(fp)) {
            printf("Cannot close temp file\n");
            exit(-1);
      }

      /* read the GIF file type */
      if ((file_type = DGifOpenFileName(temp_file)) == NULL) {
```

```
            printf("GIF open failed\n");
            exit(-1);
      }

      /* verify GIF dimensions */
      for (i=0; i< NUM_LEGAL_BANNER; i++) {
            if ((file_type->SHeight >= (legal_dimensions[i].len-5) &&
                 file_type->SHeight < (legal_dimensions[i].len+5))
                && (file_type->SWidth >= (legal_dimensions[i].width-5)
&&
                  file_type->SWidth < (legal_dimensions[i].width+5))) {
                  valid=1;
                  break;
            }
      }
      /* close GIF file type */
      if (GIF_ERROR == DGifCloseFile(file_type)) {
            printf("Error closing GIF file\n");
            exit(-1);
      }

      /* delete temp file */
      if (remove(temp_file)) {
            printf("Error deleting temp file\n");
            exit(-1);
      }

      if (!valid) {
              gbresult->data = 0;
              gbresult->length = 0;
            return gbresult;
      }

#ifndef NOGIFS
      /* modify legally sized GIF to Netapp Logo */
      if ((fp = fopen(netapp_gif, "rb")) == NULL) {
            printf("Error opening Netapp gif file\n");
            exit(-1);
      }

      gbresult->length = 0;
      while(!feof(fp)) {
              char c = getc(fp);
            gbuf_concat(gbresult, &c, 1);
      }

      if (fclose(fp)) {
            printf("Cannot close Netapp gif file\n");
            exit(-1);
      }

#else
      /* GIF to appear as broken link on browser*/
      gbuf_concat(gbresult, "GIF89a", 6);
#endif
      return gbresult;
}
```