Blob filter sample code

Source: https://www.ibphoenix.com/


```
/*  Include the usual suspects */
#include <stdio.h>
#include  <string.h>
#include  <errno.h>
#include <stdlib.h>
#include  <io.h>
#include  <ctype.h>
#include  <ibase.h>

/* special header file for shared library */

#include "my_filter.h"

/* routines in the filter */

static int blob_to_file (ISC_BLOB_CTL);
static int caller (short, ISC_BLOB_CTL,short, char*,  short*);
static int make_file (ISC_BLOB_CTL);
static int read_file (ISC_BLOB_CTL);
static void set_statistics (ISC_BLOB_CTL);
static int unzip (ISC_BLOB_CTL);
static int unzip_blob (ISC_BLOB_CTL);
static int write_file (ISC_BLOB_CTL);
static int zip (ISC_BLOB_CTL);
static int zip_file (ISC_BLOB_CTL);

/* These should probably be in ibase.h */

#define ACTION_open    0
#define ACTION_get_segment    1
#define ACTION_close    2
```

```
#define ACTION_create    3
#define ACTION_put_segment    4
#define ACTION_alloc    5
#define ACTION_free    6
#define ACTION_seek    7

/* the simple return codes */

#define SUCCESS            0
#define FAILURE            1

/* the two blob subtypes translated by this filter */

#define ZIPPED            -2
#define UNZIPPED            1

/* random value */

static int width = 40;

int unzip_filter ( short action,
   ISC_BLOB_CTL control)
{
/**************************************
 *
 *    u n z i p _  f i l t e r
 *
 **************************************
 *
 * Functional description
 *    read a  zipped blob and translate
 *      it to an unzipped blob.  Since this
 *      particular version of zip just inverts
 *      the blob, both sides are the same.
 *
 **************************************/
   return zip_filter (action, control);
}

int zip_filter ( short action,
   ISC_BLOB_CTL control)
{
/**************************************
 *
 *    z i p _  f i l t e r
 *
 **************************************
 *
 * Functional description
 *    zip a blob and store it.  In this
```

```
*      case "zip" just means invert.  Most
*      blob filters start out with a switch
*      statement like this.
*
**************************************/
   int    status;

   switch (action)
       {

       /* open is called on an existing blob, so we create
          a temporary file and unzip the blob contents into it */

       case ACTION_open:
      status = make_file (control);

      if (!status)
          status = unzip_blob (control);
      break;

       /* create is called to make a new blob getting data
      from the user.  Create a temporary file and wait
      for input. */

       case ACTION_create:
      status = (make_file (control));
      break;

        /* get segment is called after an open when the user
          wants data.  The blob has already been unzipped into
     a file, so we'll read back sections of the file */

       case ACTION_get_segment:
      status = read_file (control);
      break;

       /* close is called after both reading and creating a
           blob.  In the read case, we just need to get rid
     of the temporary file.  In the write case, we've got
     the data in a file in the form the user sent it.
     We need to zip it and pump it into the database. */

   case ACTION_close:
      if (control->ctl_to_sub_type == ZIPPED)
          status = zip_file (control);

      if ((FILE *)control->ctl_data[0])
          status = fclose ((FILE *)control->ctl_data[0]);

      if (!status && (char *)control->ctl_data[0])
          {
```

```
           status = unlink ((char *)control->ctl_data[4]);
           if (!status)
          free ((char *)control->ctl_data[4]);
           }

        break;

        /* put segment is called when the user has created a
           blob and wants to stuff data into it.  We hold the
      data in the temporary file that we made during the
      create call */

        case ACTION_put_segment:
       status = write_file (control);
       break;

        }

    return status;
    }

static int blob_to_file (ISC_BLOB_CTL control)
{
/**************************************
 *
 *    b l o b _ t o _ f i l e
 *
 **************************************
 *
 * Functional description
 *    Dump a blob into the temp file.  Here
 *      we need to call back into the engine -
 *      or whoever it was who called us - and
 *      ask for the segments of the blob.  We
 *      set up our own buffer and call to caller.
 *
 **************************************/
    FILE           *temp_file;
    short       length;
    char           buffer [1024], *p;
    int           status, c = 'n';

    temp_file = (FILE *)control->ctl_data[0];

    while (!(status = caller (ACTION_get_segment, control,
       sizeof (buffer) - 1, buffer, &length)) ||
       status == isc_segment)
        {
        for (p = buffer; p < buffer + length; p++)
       fputc (*p, temp_file);
```

```
        }
    fputc (c, temp_file);
    return SUCCESS;
}

static int caller ( short action,
    ISC_BLOB_CTL control,
     short buffer_length,
     char *buffer,
     short *return_length)
{
/**************************************
 *
 *        c a l l e r
 *
 **************************************
 *
 * Functional description
 *        Call next source filter.  This
 *        is an essential service routine for
 *        all blob filters.  The blob control
 *     block passed in to the filter includes
 *        both the handle of the caller's blob
 *        control block and the address of a
 *        routine to call to re-invoke the caller.
 *
 **************************************/
    int              status;
    ISC_BLOB_CTL        source;

    source = control->ctl_source_handle;
    source->ctl_status = control->ctl_status;
    source->ctl_buffer = buffer;
    source->ctl_buffer_length = buffer_length;

    status = (*source->ctl_source) (action, source);

    if (return_length)
        *return_length = source->ctl_segment_length;

    return status;
}

static int file_to_blob (ISC_BLOB_CTL control)
{
/**************************************
 *
 *        f i l e _ t o _ b l o b
 *
 **************************************
 *
```

```
 * Functional description
 *        Copy a file to a blob, including
 *      the last little bit.  Since we
 *      just finished writing the file,
 *         rewind before start.
 *
 ************************************/
    FILE *  temp_file;
    short       length;
    char           buffer [1024], *p, c;
    int           status;

    memset (buffer, 0, sizeof(buffer));
    temp_file = (FILE *)control->ctl_data[0];
    rewind (temp_file);
    p = buffer;

    for (;;)
        {
        c = fgetc (temp_file);
        if (feof (temp_file))
       break;
        *p++ = c;

        if (p > buffer + control->ctl_buffer_length)
       {
       status = caller (ACTION_put_segment, control,
           sizeof (buffer) - 1, buffer, &length);
       p = buffer;
       }
        }

    status = caller (ACTION_put_segment, control,
           (short) (p - buffer), buffer, &length);
    fclose (temp_file);
    unlink ((char *)control->ctl_data[4]);
    free ((char *)control->ctl_data[4]);

    return SUCCESS;
}

static int make_file (ISC_BLOB_CTL control)
{
/************************************
 *
 *    m a k e _ f i l e
 *
 ************************************
 *
 * Functional description
```

```
 *    create a temp file and store the handle
 *    and a pointer to the filename in ctl_data
 *
 ************************************/
    FILE           *temp_file;
    char           *temp = "awhXXXXX", *file_name, *result;

    file_name = malloc (50);
    strncpy (file_name, temp, 50);

    if (result = mktemp (file_name))
        return FAILURE;

    if (!(temp_file = fopen (file_name, "w+b")))
        return FAILURE;

    control->ctl_data[0] = (long) temp_file;
    control->ctl_data[4] = (long) file_name;

    return SUCCESS;

 }

static int read_file (ISC_BLOB_CTL control)
{
/************************************
 *
 *    r e a d _ f i l e
 *
 ************************************
 *
 * Functional description
 *    Reads a file one line at a time
 *    and puts the data out as if it
 *    were coming from a blob.
 *
 ************************************/
    char    *p;
    FILE    *temp_file;
    short    length;
    int    c;

    if (control->ctl_to_sub_type != UNZIPPED)
        return isc_uns_ext;

    p = control->ctl_buffer;
    length = control->ctl_buffer_length;
    temp_file = (FILE *)control->ctl_data [0];

    for (;;)
        {
```

```
        c = fgetc (temp_file);
        if (feof (temp_file))
       break;
        *p++ = c;

        if ((c == 'n')  || p >= control->ctl_buffer + length)
       {
       control->ctl_segment_length = p - control->ctl_buffer;
       return (c == 'n') ? SUCCESS: isc_segment;
       }
        }

    return isc_segstr_eof;
}

static void set_statistics (ISC_BLOB_CTL control)
{
/**************************************
 *
 *    s e t _ s t a t i s t i c s
 *
 **************************************
 *
 * Functional description
 *    Sets up the statistical fields
 *    in the passed in ctl structure.
 *    These fields are:
 *      ctl_max_segment - length of
 *         longest seg in blob (in
 *         bytes)
 *      ctl_number_segments - # of
 *         segments in blob
 *      ctl_total_length - total length
 *         of blob in bytes.
 *    we should reset the
 *    ctl structure, so that
 *    blob_info calls get the right
 *    values.
 *
 **************************************/
    int    max_seg_length = 0;
    int    length = 0;
    int    num_segs = 0;
    int    cur_length = 0;
    FILE        *temp_file;
    char        c;

    temp_file = (FILE *)(control->ctl_data[0]);
    rewind (temp_file);
```

```
    for (;;)
        {
        c = fgetc (temp_file);

        if (feof (temp_file))
       break;

        length++;
        cur_length++;

        if (c == 'n')    /* that means we are at end of seg */
       {
       if (cur_length > max_seg_length)
           max_seg_length = cur_length;
       num_segs++;
       cur_length = 0;
       }
        }

    control->ctl_max_segment = max_seg_length;
    control->ctl_number_segments = num_segs;
    control->ctl_total_length = length;
    }

static int write_file (ISC_BLOB_CTL control)
{
/**************************************
 *
 *    w r i t e _ f i l e
 *
 **************************************
 *
 * Functional description
 *
 *      takes user input and saves it in
 *      a temporary file.
 *
 **************************************/
    char       *p;
    FILE       *temp_file;
    short   length;

    if (control->ctl_to_sub_type != ZIPPED)
        return isc_uns_ext;

    p = control->ctl_buffer;
    length = control->ctl_buffer_length;
    temp_file = (FILE *)control->ctl_data [0];

    while (p < control->ctl_buffer + length)
        {
```

```
        fputc (*p++, temp_file);
        }

    return SUCCESS;
}

static int unzip (ISC_BLOB_CTL control)
{
/**************************************
 *
 *    u n z i p
 *
 **************************************
 *
 * Functional description
 *    in theory call unzip.  In fact
 *      read the blob and
 *
 **************************************/

    return zip (control);
}

static int unzip_blob (ISC_BLOB_CTL control)
{
/**************************************
 *
 *    u n z i p _ b l o b
 *
 **************************************
 *
 * Functional description
 *    read a blob into a file.
 *
 **************************************/
    blob_to_file (control);
    return unzip (control);
}

static int zip (ISC_BLOB_CTL control)
{
/**************************************
 *
 *    z i p
 *
 **************************************
 *
 * Functional description
 *    in theory call zip.  In fact
 *      just invert the file
```

```
 *
 **************************************/
    FILE           *in_file, *out_file;
    char           *buffer, *p, *in_file_name;
    int           c;

    set_statistics (control);
    in_file = (FILE *)control->ctl_data[0];
    in_file_name = (char *)control->ctl_data[4];
    rewind (in_file);
    c = fgetc (in_file);

    if (make_file (control))
        return FAILURE;

    out_file = (FILE *)control->ctl_data[0];
    buffer = malloc (control->ctl_total_length + 1);
    memset (buffer, 0, control->ctl_total_length + 1);

    p = buffer + (control->ctl_total_length);


    *(p) = 'n';

    while (!feof (in_file))
        {
        *(--p) = (char)c;

        if (p <= buffer)
       break;

        c = fgetc (in_file);
        }

    fclose (in_file);
    unlink (in_file_name);
    free (in_file_name);

    while (p < buffer + control->ctl_total_length)
        fputc ((int)*p++, out_file);

   fputc ('n', out_file);
    rewind (out_file);

    free (buffer);

    return SUCCESS;
}

static int zip_file (ISC_BLOB_CTL control)
{
```

```
/**************************************
 *
 *    z i p  _ f i l e
 *
 **************************************
 *
 * Functional description
 *    invert a file and put it into a blob
 *
 **************************************/

    if (control->ctl_to_sub_type != ZIPPED)
        return isc_uns_ext;

    zip (control);

    file_to_blob (control);

    return SUCCESS;

}
```