Subject: :(errors on sfx project
Posted by Rishi on Tue, 04 Jan 2011 05:49:51 GMT
View Forum Message <> Reply to Message

/* zpsfx.cpp v1.00 - ZPAQ self extracting stub.
  Written by Matt Mahoney, Oct. 20, 2010.

zpsfx is placed in the public domain. It may be used without
restriction. It is provided "as is" with no warranty.

This program is intended to be as simple as possible to illustrate how
to create self extracting ZPAQ archives. You can customize it.
To compile this program:

  g++ -O2 -s zpsfx.cpp libzpaq.cpp libzpaqo.cpp -DNDEBUG
  upx a.exe

Or use appropriate optimization options. -s and upx can be used to make
the executable smaller. -DNDEBUG turns off run time checks.
To convert a ZPAQ archive to a self extracting archive:

  copy/b zpsfx.exe+zpsfx.tag+archive.zpaq archive.exe

zpsfx.tag is a 13 byte file used to mark the start of the compressed data
that is appended. Alternatively, zpaq with the "t" modifier will append
the same tag ("a" appends).

  copy zpsfx.exe archive.exe
  zpaq ta archive.exe files...

To extract:

  archive.exe

The program reads itself and decompresses the appended archive.
You must enter the .exe extension as shown. If the file is not in
the current folder then you need to specify the path. The PATH environment
variable won't be used to find it.

Files will be extracted and named according to the stored filenames. It will
overwrite existing files without warning. It will not create directories.
It will exit if any errors occur, such as the first segment not having a
stored filename. SHA-1 checksums are not verified.
*/

#include "libzpaq.h"

```
#include <stdio.h>
#include <stdlib.h>

// An error handler is required as shown in this example. libzpaq will
// call it with an English language message in case of a fatal error.

namespace libzpaq {
  void error(const char* msg) {
    fprintf(stderr, "%s\n", msg);
    exit(1);
  }

  // Default models. If you compile with -DNOOPT and not link with libzpaqo.cpp
  // then zpsfx.exe will be a few KB smaller. However, if the archive was
  // compressed with any of the three default models (fast, mid, max), then
  // decompression would not be as fast.
#ifdef NOOPT
  const char models[]={0,0};
  void ZPAQL::run(U32 a) {run0(a);}
  int Predictor::predict() {return predict0();}
  void Predictor::update(int y) {update0(y);}
#endif
}

// I/O types. A Decompresser reads from a Reader and writes to a Writer,
// which are abstract base classes.
// The user must override Reader::get() and Writer::put() to read and write
// bytes or EOF (-1) as in this example. This allows decompression (and
// compression) to/from either files or data structures in memory.

struct File: public libzpaq::Reader, public libzpaq::Writer {
  FILE* f;
  File(FILE* f_): f(f_) {}
  int get() {return getc(f);}
  void put(int c) {putc(c, f);}
};

// The Decompresser will read a filename from the archive and write it here.
// Again we override Writer::put(), but this time it appends a byte to
// a 0 terminated string until it is full. There is no limit on the filename
// size in the archive, so this will fail if the filename is huge.

struct Buf: public libzpaq::Writer {
  enum {SIZE=511}; // max length of output
  int len;         // length of output in s
  char s[SIZE+1];  // 0 termimated string
  void clear() {s[len=0]=0;}  // erase contents
  Buf() {clear();}
```

```
  void put(int c) {if (len<SIZE) s[len]=c, s[++len]=0;}
};

// Extract ZPAQ compressed data appended to this program executable
int mai(char** argv) {

  // Find self. You could also try adding a .exe extension and searching
  // the PATH, but we will keep it simple.
  File in(fopen(argv[0], "rb"));
  if (!in.f) perror(argv[0]), exit(1);

  // Extract each named segment to a separate file. The first segment must
  // be named. Unnamed segments are appended to the previous segment.
  libzpaq::Decompresser d;
  d.setInput(&in);
  File out(0);

  // Read all blocks.
  // The marker tag (zpsfx.tag) is used to find the first block.
  while (d.findBlock()) {

    // Read all segments in the block.
    // The filename is read from the segment header.
    Buf filename;
    while (d.findFilename(&filename)) {
      printf("Extracting %s\n", filename.s);

      // If the segment is named then open a new output file.
      if (filename.len) {  // segment is named?
        if (out.f) fclose(out.f);
        out.f=fopen(filename.s, "wb");
      }
      if (!out.f) perror(filename.s), exit(1);
      d.setOutput(&out);
      filename.clear();

      // Ignore comment after filename from the segment header.
      d.readComment();

      // Decompress up to the end of the segment.
      d.decompress();

      // Ignore SHA-1 checksum if any.
      d.readSegmentEnd();
    }
  }
  if (out.f) fclose(out.f);
  return 0;
```

```
}

#include <CtrlLib/CtrlLib.h>

// http://java.sun.com/docs/books/tutorial/uiswing/start/swingTour.html

using namespace Upp;

struct ButtonApp : TopWindow {
   int    count;
   Button button;
   Label  label;

   void RefreshLabel()
   {
      label = Format("Number of button clicks %d", count);
   }
   void Click()
   {
      mai(argv);
   }

   typedef ButtonApp CLASSNAME;

   ButtonApp()
   {
      count = 0;
      button <<= THISBACK(Click);
      button.SetLabel("&I'm an Ultimate++ button!");
      Add(button.VCenterPos(20).HCenterPos(200));
      Add(label.BottomPos(0, 20).HCenterPos(200));
      label.SetAlign(ALIGN_CENTER);
      Sizeable().Zoomable();
      RefreshLabel();
   }
};

int main(int argc)
{
   ButtonApp();
}
```

I want to pass argv to mai(). it should run called by a button. how?