Include .png in executable

user name (23) May 25, 2009 at 2:12am UTC

Is there a way to bundle a PNG image (or any other file) into the executable itself, instead of having to distribute 2 separate files? I'd like to use that image as a kind of background of a window. I know that this isn't really a programming question but any help would still be appreciated.

I am running Linux with GCC and wxWidgets installed.

Disch (527) May 25, 2009 at 2:12am UTC

The way I do it is pretty ghetto, but it works:

I whipped up a program which reads a binary file and converts it to a C++ style header file:

```
1 const size_t WHATEVER_FILE_SIZE = 0x1234;
2
3 const uint8_t WHATEVER_FILE[] = {
4 0x01,0x02
5 ...
6 };
```

Just #include that file somewhere (probably only do it from one source file) and instead of reading from a file, read from that memory buffer. wx provides a way to do that easily (wxMemoryInputStream I think -- something like that)

helios (2791) May 25, 2009 at 2:12am UTC

One word of caution: depending on the size of the file, using this technique may raise both binary size and compilation time considerably.

Also, the array is probably best left as global and defined inside a .cpp, but declared in headers, so it will only need to be compiled once. Example:

```
1 //definition (in a single .cpp):
2 //Note: uint8_t is not standard.
3 const char file[]={/*...*/};
4 //definition (in every file that uses it or in a single .h):
5 extern const char file[];
```

Last edited on May 25, 2009 at 2:12am UTC

user name (23) May 25, 2009 at 2:12am UTC

OK, thank you!

user name (23) May 25, 2009 at 2:12am UTC

I just finished making the program that creates header files from binaries using `ifstream` and `get()` (thanks Disch!), and everything works fine. I compiled a basic hello world application, used the program and generated an array, put it in wxMemoryInputStream and wrote it to a wxFileOutputStream to my folder, so theoretically I should have made an identical copy of the hello world program. However when I tried to run it, there was an error that just said

```
$ ./hello
Killed
```

I used:

```
$ cmp ./hello ./Desktop/asdf
./hello ./Desktop/asdf differ: byte 25, line 1
```

I opened up a hex editor and found that everywhere the original had `0x80`, the new one has `0xFF`. What went wrong? (I can provide the source code for the header file generator if you want, but I doubt that's the problem -- I'm suspecting ifstream::get() is not working)

helios (2791) May 25, 2009 at 2:12am UTC

If I had to guess, I'd say that, in the program that generates the array, you either a) opened the file as text instead of binary, or b) used a signed type.
Post the source and I'll be able to tell you more precisely.

user name (23)

Well, I tried opening the file as a binary instead, but it still doesn't work... Here's the source:

```
1 // bin2h.cpp
2
3 #include "bin2h.h"
4
5 BinHeader::BinHeader(string outputLoc)
6 {
```

```
    if (outputLoc != "") OpenHeader(outputLoc);
    else isInit = false;
}

bool BinHeader::OpenHeader(string outputLoc)
{
    if (isInit) fHeader.close();
    fHeader.open(outputLoc.c_str(), ios::binary | ios::in);
    return (isInit = fHeader.is_open());
}

bool BinHeader::AddFile(string fileLoc, string arrName)
{
    if (isInit) {
        if (CheckArrName(arrName)) {
            ifstream fFile(fileLoc.c_str());
            if (fFile.is_open()) {
                int i = 0;
                char chr;
                fHeader << "const char " << arrName << "[]={";
                fFile.get(chr);
                if (fFile.good())
                    while (true) {
                        fHeader << CharHex(chr);
                        i++;
                        fFile.get(chr);
                        if (!fFile.good()) break;
                        else fHeader << ",";
                    }
                fHeader << "};\nconst long " << arrName << "_len=" << i << ";" << endl;
                binList[0].push_back(arrName);
                binList[1].push_back(fileLoc);
                return true;
            }
        }
    }
    return false;
}

bool BinHeader::CheckArrName(string arrName)
{
    return (CheckArrNameValid(arrName) && find(binList[0].begin(), binList[0].end(), arrName.c_str()) == binList[0].end() && find(binLi
}

void BinHeader::CloseHeader()
{
    if (isInit) {
        fHeader.close();
        isInit = false;
    }
}

BinHeader::~BinHeader()
{
    CloseHeader();
}

string CharHex(char chr)
{
    char tmp[5];
    snprintf(tmp, 5, "0x%02X, ", chr);
    return string(tmp);
}

bool CheckArrNameValid(string arrName)
{
    return (isalnum(arrName.at(0)) && arrName.substr(1).find_first_not_of(VALID_CHARS) == string::npos);
}

int main(int argc, char* argv[])
{
    BinHeader bh;
    string hfl, tmp;
    vector<string> vf[2];
    ostringstream stmp;
    do {
        cout << "Create header file at: ";
        getline(cin, hfl);
    } while (hfl == "");
    while (true) {
        cout << "Include file: ";
        getline(cin, tmp);
        if (!(tmp == "")) {
            vf[0].push_back(tmp);
            cout << "Array name: ";
            getline(cin, tmp);
            if (!bh.CheckArrName(tmp)) {
                cout << "That name is invalid or has been taken." << endl;
                vf[0].pop_back();
            }
            else vf[1].push_back(tmp);
        }
        cout << "Add another file? (y/N) ";
        getline(cin, tmp);
        if (tmp == "" || toupper(tmp.at(0)) == 'N') break;
    }
    cout << "==PROGRESS==\nCreating header file...";
    if (!bh.OpenHeader(hfl)) {
        cout << "\nERROR: Could not create header file." << endl;
```

```
106         return 1;
107     }
108     for (int i = 0; i < vf[0].size(); i++) {
109         cout << "\nAdding file '" << vf[0].at(i) << "' ...";
110         stmp << "bin" << i;
111         if (!bh.AddFile(vf[0].at(i), (vf[1].at(i) == "" ? stmp.str().c_str() : vf[1].at(i))))
112             cout << "\nERROR: Could not read file '" << vf[0].at(i) << "'.";
113         stmp.seekp(0);
114     }
115     cout << endl;
116     return 0;
117 }
```

```
1  // bin2h.h
2  #include <cstdio>
3  #include <cstring>
4  #include <iostream>
5  #include <fstream>
6  #include <sstream>
7  #include <vector>
8  #include <algorithm>
9  #include <cctype>
10
11 using namespace std;
12
13 class BinHeader {
14     ofstream fHeader;
15     bool isInit;
16     vector<string> binList[2];
17 public:
18     BinHeader(string outputLoc = "");
19     bool OpenHeader(string outputLoc);
20     bool AddFile(string fileLoc, string arrName = "");
21     bool CheckArrName(string arrName);
22     void CloseHeader();
23     ~BinHeader();
24 };
25
26 string CharHex(char chr);
27
28 bool CheckArrNameValid(string arrName);
29 #define VALID_CHARS        "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789_"
```

Also any improvements, bugs spotted, coding style feedback etc. would be welcome.

---

helios (2791) May 25, 2009 at 2:12am UTC

Okay, see, this is completely fucked up.

bin2h.cpp:
Line 7: You're ignoring the return value of OpenHeader(). The file may not have opened.
Line 14: You're opening an std::ofstream with the std::in flag. You're also opening as binary, which doesn't really make sense, since you want to write a C++ header, and those are always text. I meant you needed to open the input file as binary. I didn't think I had to make it explicit I wasn't talking about the output file.
Line 22: *This* is the file you should be opening as binary.

It seems to me like you made this program overly complicated. All it really needed to do was ask for an input file, an output file, and an array name, then write to the file accordingly. There was no need to use classes, and there definitely was no need to check that the array name had not already been used. Since *you* were going to be the only user of the program, you made it unnecessarily robust (yes, there's such a thing).

---

user name (23) May 25, 2009 at 2:12am UTC

I suppose it is a bit over-complicated...

Btw, I only added the `ios::binary | ios::in` bit when you mentioned it in an earlier post, I must have misplaced it :). However when I move it to where it's meant to be, the result is still the same. I created a file using a hex editor that contained `80 FF`, but the header file still says `{0xFF,0xFF}`...

---

helios (2791) May 25, 2009 at 2:12am UTC

You're opening as binary, but still using text methods to read the stream. Use std::istream::read().

---

user name (23) May 25, 2009 at 2:12am UTC

Is this right?

```
1 char chr;
2 // ...
3 fFile.read(&chr, 1);    // instead of fFile.get(chr);
```

If so, I've tried that and it also doesn't work.

Grey Wolf (1027) May 25, 2009 at 2:12am UTC

The following will read 100 bytes of data into a buffer:

```
1 ...
2     char buffer[100];
```

```
    ifstream myFile ("data.bin", ios::in | ios::binary);
    myFile.read (buffer, 100);
    if (!myFile)
    {
        // An error occurred!
        // myFile.gcount() returns the number of bytes read.
        // calling myFile.clear() will reset the stream state
        // so it is usable again.
    }
...
```

user name (23) May 25, 2009 at 2:12am UTC

Found the problem!

```
// readtest.cpp
#include <iostream>
#include <cstdio>
#include <fstream>
using namespace std;
int main()
{
    char buffer[2];
    ifstream myFile ("test.bin", ios::in | ios::binary);    // in ./test.bin there are two bytes, 80 followed by FF
    myFile.read(buffer, 2);
    cout << (buffer[0] == buffer[1] ? "Raw input is the same." : "Raw input is different.") << endl;
    char hexbuf[5];
    snprintf(hexbuf, 5, "0x%02X", buffer[0]);
    cout << hexbuf << ", ";
    snprintf(hexbuf, 5, "0x%02X", buffer[1]);
    cout << hexbuf << endl;
    return 0;
}
```

```
$ hex ./test.bin
0x00000000: 80 ff                  -                              ..
$ g++ ./readtest.cpp -o ./readtest
$ ./readtest
Raw input is different.
0xFF, 0xFF
$
```

It turns out that my choice of `snprintf` was wrong then... any alternatives (that work)?

Last edited on May 25, 2009 at 2:12am UTC

user name (23) May 25, 2009 at 2:12am UTC

OK, so I've got this:

```
// ... (see above posts)
string CharHex(char chr)
{
        ostringstream ss;
        ss << "0x" << hex << setw(4) << setfill('0') << (int)chr;
        return ss.str();
}
```

but it's giving me `0xffffff80` not `0x80` -- the setw manipulator only sets the minimum width. Any help would be appreciated.

Last edited on May 25, 2009 at 2:12am UTC

Disch (527) May 25, 2009 at 2:12am UTC

I know C in a C++ world is often frowned upon, but I never liked C++ iostreams -- such a pain to use. Especially for file io. Here's a quickie thing I made which uses cstdio (read: uncompiled, untested)

```
#include <cstdio>

void PrintRow(FILE* dst,const unsigned char* buf,unsigned count)
{
  fprintf(dst,"\n    ");
  while(count > 1)
  {
    fprintf(dst,"0x%02X,",*buf);
    ++buf;
    --count;
  }
  if(count > 0)
    fprintf(dst,"0x%02X",*buf);
}

void BinToCpp(
    FILE* src, // source file, opened with "rb"
    FILE* dst) // dest file, opened with "wt"   note neither file is closed by this
{
  // determine source file size
  unsigned srcsize;
  fseek(src,0,SEEK_END);
  srcsize = (unsigned)ftell(src);
  fseek(src,0,SEEK_SET);

  // dump source file size to output file
  fprintf(dst,"\n\nconst unsigned FILE_SIZE = %u;\n\n",srcsize);

```

```
29   //
30   fprintf(dst,"const unsigned char FILE_DATA[] = {");
31
32   // take the source file in 16 byte blocks
33   unsigned char buf[16];
34   while(srcsize > 16)
35   {
36     fread(buf,1,16,src);
37     PrintRow(dst,buf,16);
38     fprintf(dst,",");
39     srcsize -= 16;
40   }
41
42   // do the rest of the file
43   fread(buf,1,srcsize,src);
44   PrintRow(dst,buf,srcsize);
45   fprintf(dst,"\n};\n\n");
46 }
```

helios (2791) May 25, 2009 at 2:12am UTC

0x80 (or 10000000b) in char (not unsigned char) is -128. To convert a signed integer to a higher signed type, the biggest bit is copied to the new bits, so the same number in signed 32-bit integers is 0xFFFFFF80 (or 11111111 11111111 11111111 10000000b)
0x80 in unsigned char is 128. When converting an unsigned type to a bigger type, the new bits are set to zero, so 128 in 32-bit unsigned integers is 0x00000080 (or 00000000 00000000 00000000 10000000b).

Like I said in my second post, use unsigned types when dealing with binary data.

user name (23) May 25, 2009 at 2:12am UTC

Thank you very much!

This topic is archived - New replies not allowed.