Proposed Multiple File Packing Protocol

Harry R. Chesley

Introduction

This document describes the packed file format (protocol) used by the PackIt program. It is being presented as a possible informal standard; details on how it might work in the context of other existing standards are described later.

The goals of the PackIt protocol are:

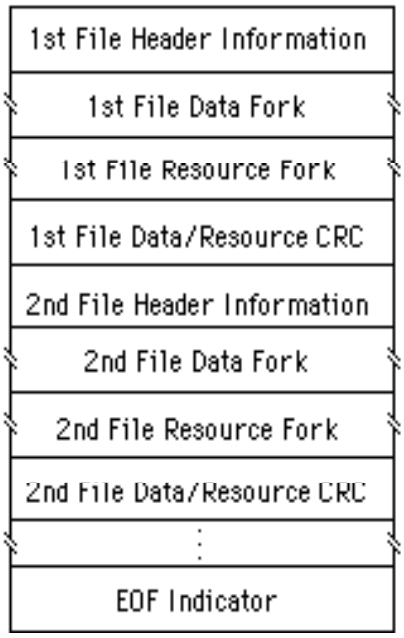
- Allow multiple files to be transported as a single file.
- Preserve both forks of the file, as well as appropriate header information.
- Provide a strong integrity verification to ensure that the files are intact, and also that unpacking them will in no way damage the file system.

Specifically discluded from PackIt's goals are:

- Transport of the files between the packing and the unpacking sites.
- Error correction.
- Data compression.

Format

The file format used by PackIt is quite straight-forward:



Each sub-file consists of a block of header information, the contents of the file's data fork, the resource fork, and a CRC over the data and resource forks. Following all of the sub-files is an end-of-file indicator.

The header information has the following format:

```
struct packFHead {          /* Packed file info header format. */
        long magic;                 /* Magic number. */
        char fName[64];             /* Suggested file name. */
        OSType fType;               /* File type. */
        OSType fCreator;            /* File creator. */
        short fFlags;       /* File (finder) flags. */
        int fLocked;                /* File locked if non-zero. */
        long DSize;                 /* Data fork size (bytes). */
        long RSize;                 /* Resource fork size (bytes). */
        long fCrDate;       /* File creation date. */
        long fModDate;              /* File modification date. */
        unsigned hCrc;              /* Header CRC. */
};
```

The magic number is 'PMag'. This should also be thought of as a sub-file format type; it may be used in the future to identify a different packing scheme, allowing different packing schemes to be used within the same packed file. The name, type, creator, finder flags, locked flag, data fork size, resource fork size, creation date, and modification date are all the standard information used in other file transfer protocols. Although all of the finder flag bits are sent, the current implementation masks off all but "invisible" and "has bundle" when unpacking.

The CRC is taken across the entire header, including the magic number. It allows the unpacker to verify the header before attempting to create a file using that information. The CRC algorithm is given later.

The EOF indicator is 'PEnd'. It allows the unpacker to verify that all files were received, that the file was not truncated during transmission. Most unpackers will want to allow unpacking of the part that did come across in this event, but they should notify the user that not everything is there. This can be especially important with programs that depend on accompanying data files.

The CRC is generated (and checked) using the following algorithm (it's CRC-CCITT):

```
/*      Return the CRC across each byte in the block pointed to by blk, of size sz. Start with
        oldcrc, so we can use it across multiple blocks.
*/

crc(oldcrc,blk,sz)

unsigned oldcrc;
char *blk;
long sz;

{
        register unsigned crcret;
        register char *ptr;
        register long cnt;
        register unsigned i;

        crcret = oldcrc; ptr = blk; cnt = sz;
```

```
        while (cnt--) {
                crcret = crcret ^ ( ((int) *ptr++) << 8);
                for (i = 0; i < 8; i++)
                        if (crcret & 0x8000) crcret = (crcret<<1) ^ 0x1021;
                        else crcret <<= 1;
        };

        return(crcret);
}
```

This algorithm is acceptably fast, but it's fairly simple to produce a faster, table-driven version.

Using a CRC checks that the data survived transport regardless of how many hops were involved and no matter how poor the error detection and correction of any of those hops. I believe this to be especially important in this sort of program since corruption of the contents of the file can lead to executing random data or creating a file with incorrect header information. The consequences can be damaged file systems, confused users, and maybe even lost data.

PackIt and MacBinary

It's quite possible for the PackIt and MacBinary protocols to live completely separately. In this case, a group of files is packed with PackIt, then transported with MacBinary (or other protocols), then unpacked. This sort of separation has been accepted on other machines. It does make a two step process out of file transfer, however, so I do tend to agree with others that combining the two protocols would be a step forward.

I have two caveats: (1) Real-time unpacking makes it difficult for the user to be given a choice during the unpacking process — the current PackIt program suggests a file name but does not require the user to use it. And (2) with a two step process, a number of extensions to PackIt immediately suggest themselves, including data compression, encryption, etc.; this rich a set of transformations will be very difficult to standardize, probably impossible. Neither of these is insurmountable.

If you consider the PackIt protocol as described above to be a new version of MacBinary, there are two ways in which it can be reconciled with the current MacBinary. The first is to add a MacBinary version byte at the start of the PackIt protocol. The other approach is to use the initial 'P' in the 'PMag' magic number as the version. In other words, PackIt would be MacBinary version 80.

Additional Features

The one area where I'd like to see PackIt improved is giving the unpacking program the ability to handle files with garbage prepended to them. This can happen quite easily during transport, and recovering from it would be a big plus.

Because of the header CRC, however, this can be added without changing the protocol. If the unpacker scans the file for the sequence 'PMag', it can verify that what it finds is a valid header by checking the CRC. This allows it to maintain a high level of confidence in the integrity of the header despite leading gibberish.