[Q]: Как прикрутить к файлу расширенный атрибут - исходник

[A]: Dmitry Zavalishin (2:5020/32)

Елки-палки, как долго я боялся за это браться. Оказалось, если не лезть в дебри, то все вполне терпимо.

Разъяснения:

static bool set_ea(const char *file_name, const char *ea_name, const char *ea_data, int ea_data_len);

Берет и втыкает в file_name расширенный атрибут по имени ea_name, стирая старый полностью. Значение (двоичное) берется из ea_data, длина его в байтах - из ea_data_len.

bool set_ea_ASCII(const char *fn, const char* ea_name, string data);

Кодирует строку data в соответствии с правилами полуоси и засовывает результат в указаный EA соответствующего файла. Это пригодно для EA типа “.SUBJECT”, “.LONGNAME”.

bool set_ea_MVMT_ASCII(const char *fn, const char* ea_name, vector <string> data);

Кодирует группу строк как мультитиповый мультиэлементный EA и пристегивает его к файлу. Это пригодно для EA типа “.HISTORY”, “.COMMENTS”. Вообще тут пошел бы и MVST, но, говорят, традиционно используется MVMT.

bool set_ea_MVST_ASCII(const char *fn, const char* ea_name, vector <string> data);

Кодирует группу строк как однотиповый мультиэлементный EA и пристегивает его к файлу. Это пригодно для EA типа “.KEYPHRASES”. Эту функцию на данный момент я даже не проверял в работе, так что если что - извините.

/*\ * The software included, file formats and basic algorithms are * copyright (C) 1995,96 by Dmitry Zavalishin. All rights reserved. * * Module: OS/2 EAs * * $Log: ea.C $ * Revision 1.1 1996/07/22 02:48:05 dz * Initial revision * * * * */

#include “frip.h” *#include “ea.h” #ifdef* OS2 *#define INCL_DOSFILEMGR /* File Manager values */ #define INCL_DOSERRORS /* DOS error values */ #include <os2.h>* #include <stdio.h> *#include <string.h> #pragma pack(4) static bool set_ea(const char *file_name, const char *ea_name, const char *ea_data, int ea_data_len) { APIRET rc = NO_ERROR; /* Return code */ EAOP2 op; char * databuf = new char[(64*2*1024)+1024];* twice 64K for EA data + 1024 for any case

```
  op.fpGEA2List = (PGEA2LIST)0;
  op.fpFEA2List = (PFEA2LIST)databuf;
  //char *attname = ".SUBJECT";
  int  ea_name_len = strlen( ea_name );
  if( ea_name_len > 255 )
      {
      Error("EA name too long: "+string(ea_name));
      return Err;
      }
  //char datname[] = "\xFD\xFF\x14\x00More Stupid Subject!\x0"; // FFFD, 2-
byte len, text
```

```
  //char  datlen = sizeof( datname );
  char *databufp = databuf + sizeof(long);
  *((long*)databufp) = 0; // Next field offset is zero - just one field here
  databufp += sizeof(long);
  *databufp++ = 0; // not critical
  *databufp++ = (char)ea_name_len;
  *((short*)databufp) = ea_data_len;
  databufp += sizeof(short);
  memcpy( databufp, ea_name, ea_name_len+1 ); // with trailing zero
  databufp += ea_name_len+1;
  memcpy( databufp, ea_data, ea_data_len ); // with trailing zero
  databufp += ea_data_len;
  *((long*)databuf) = databufp-databuf; // Size of all that stuff
  rc = DosSetPathInfo( file_name, FIL_QUERYEASIZE, &op, sizeof(op), 0);
  if (rc != NO_ERROR)
      {
      Error("DosSetPathInfo error");
      return Err;
      }
  delete [] databuf;
  return Ok;
  }
```

*#endif OS2 class binbuf { public: char *b; binbuf(int size) { b = new char[size]; } ~binbuf() { delete [] b; } }; bool set_ea_ASCII(const char *fn, const char* ea_name, string data) { #ifdef OS2 binbuf b(64*1024); char *buf = b.b; *((short*)buf) = EAT_ASCII; buf += sizeof(short); *((short*)buf) = data.length(); buf += sizeof(short); strcpy(buf, data.c_str()); return set_ea(fn, ea_name, b.b, data.length() + 4); #else OS2*

```
  return Ok;
```

*#endif OS2 } bool set_ea_MVMT_ASCII(const char *fn, const char* ea_name, vector <string> data) { #ifdef OS2 binbuf b(64*1024); char *buf = b.b; *((short*)buf) = EAT_MVMT; buf += sizeof(short);*
Default CodePage == 0

- ((short*)buf) = 0; buf += sizeof(short);
- ((short*)buf) = data.size(); buf += sizeof(short);

```
  int len = data.size();
  for( int i = 0; i < len; i++ )
      {
```

```
      if( (64*1024-1) < ((buf-b.b) + data[i].length() + 4) )
          {
          Error("vector too big to fit in EA, cut it off :(");
          break;
          }
      *((short*)buf) = EAT_ASCII;          buf += sizeof(short);
      *((short*)buf) = data[i].length();   buf += sizeof(short);
      strcpy( buf, data[i].c_str() );      buf += data[i].length();
      }
```

```
  return set_ea( fn, ea_name, b.b, buf-b.b );
```

#else OS2 *return Ok; #endif* OS2

```
  }
```

bool set_ea_MVST_ASCII(const char *fn, const char* ea_name, vector <string> data)

```
  {
```

#ifdef OS2

```
  binbuf b(64*1024);
  char *buf = b.b;
  *((short*)buf) = EAT_MVST;       buf += sizeof(short);
    // Default CodePage == 0
  *((short*)buf) = 0;              buf += sizeof(short);
  *((short*)buf) = data.size();    buf += sizeof(short);
  *((short*)buf) = EAT_ASCII;      buf += sizeof(short);
  int len = data.size();
  for( int i = 0; i < len; i++ )
      {
      if( (64*1024-1) < ((buf-b.b) + data[i].length() + 4) )
          {
          Error("vector too big to fit in EA, cut it off :(");
          break;
          }
      *((short*)buf) = data[i].length();   buf += sizeof(short);
      strcpy( buf, data[i].c_str() );      buf += data[i].length();
      }
  return set_ea( fn, ea_name, b.b, buf-b.b );
```

#else OS2 *return Ok; #endif* OS2

```
  }
```