NAME

ESPS Feature File − (.fea)

SYNOPSIS

#include <esps/esps.h>
#include <esps/fea.h>

DESCRIPTION

An ESPS Feature File consists of a header followed by a sequence of feature records. Each record consists of a number of named ‘‘fields’’ that may hold numeric, ‘‘coded’’, or (eventually) bit data or arrays of such data. (The support for bit data is not yet available.) A field is characterized by its name, data type, size (number of items), and array dimensions. The header contains a list of field names and associated information. Each feature record contains, for each field, storage sufficient to hold the given number of items of the given data type.

The numeric types include integer and floating types of various sizes, together with corresponding complex types. Each complex type has a real and an imaginary part of the corresponding integer or floating type. The real numeric types are *double*, *float*, *long*, *short*, and *“byte”* (*signed char*). Variables of the complex types may be declared with the respective typedef names *double_cplx*, *float_cplx*, *long_cplx*, *short_cplx*, and *byte_cplx* defined in the include file *esps/esps.h*. For example the definition of *short_cplx* is

```
typedef struct {short    real, imag;}  short_cplx;
```

If *x* is a variable of that type, the real and imaginary parts are accessible as *x.real* and *x.imag*. For backwards compatibility, COMPLEX is accepted as a synonym for double_cplx.

The initial support for BIT data will be as described in this paragraph. Each item of the ‘‘bit’’ type has one of two values: 0 and 1. In memory, each item of type BIT occupies one byte. In the external file, a more compact representation is used: each field of type BIT is packed, 8 bits to the byte, into the smallest number of bytes that will hold it. Any padding required to make an integer number of bytes is added at the end of the field; there is no internal padding to align rows of a multidimensional BIT field. The functions that read and write FEA records (see *get_fea_rec*(ESPS-3u) and *put_fea_rec*(ESPS-3u)) unpack and pack BIT fields automatically. Support for representing BIT fields in packed form in memory may be added later.

Fields of type *char* are available for storing character data.

A ‘‘coded’’ data type is like a C enum type; the header contains a list of possible values, which are strings, and in a record a string is represented by the short integer that gives its position in the list. A typical use for a coded type is to represent class identifiers. For example the header could associate the field name "voicing" and the set of values {"voiced", "unvoiced", "silent"} with a record item; the size would be 1 for a single item. Then in each record a value 0, 1, or 2 for that item would indicate a value of "voiced", "unvoiced", or "silent", respectively, for the voicing class of the record.

For many applications, each feature record refers to a frame of speech in some sampled-data file or, more generally, to a contiguous segment of records in some ESPS file. There is provision for flagging a feature file as ‘‘segment_labeled’’, which implies that, in addition to whatever other information is stored, each record identifies an ESPS file and a starting record and number of records in that file. A support function (*set_seg_lab*(3−ESPSu)) is provided for setting up fields to hold this information.

The header has the following layout as defined by <*esps/header.h*>. The data items common to all ESPS data files are described in *ESPS*(5−ESPS). The type-specific header structure for FEA files is shown below.

```
/* Feature File specific header */

struct fea_header {
    short           fea_type;           /* indicates special feature-file types */
    short           segment_labeled;    /* If YES, records contain file name,
                                                start & length of segment */
    unsigned short  field_count;        /* number of fields  */
    short           field_order;        /* YES if file is in field order fmt */
```

```
    char        **names;              /* name of each field */
    long        *sizes;               /* total number of items in field */
    short       *ranks;               /* number of dimensions in field */
    long        **dimens;             /* array dimensions for field */
    short       *types;               /* type (DOUBLE, FLOAT, etc.) of field */
    char        ***enums;             /* arrays of values for coded types */
    long        *starts;              /* starting point for this field */
    short       *derived;             /*indicates whether field was derived */
    char        ***srcfields;         /*for derived fields,
                                            arrays of source field names */
    short       spares[FEA_SPARES];   /* spares */
    long        ndouble;              /* number of doubles in feature record */
    long        ndcplx;               /* number of double complex in feature record */
    long        nfloat;               /* number of floats in feature record */
    long        nfcplx;               /* number of float complex in feature record */
    long        nlong;                /* number of longs in feature record */
    long        nlcplx;               /* number of long complex in feature record */
    long        nshort;               /* number of shorts in feature record */
    long        nscplx;               /* number of short complex in feature record */
    long        nbyte;                /* number of bytes in feature record */
    long        nbcplx;               /* number of byte complex in feature record */
};
```

The following items are all in the feature-file-specific header structure.

fea_type

This item may be used to indicate that the feature file is of a special type. It might imply specific uses of some of the spares, for instance, or that certain conditions apply to the set of fields that may or must be defined. FEA_GEN indicates a completely general feature file type with no special conditions except those implied by *segment_labeled.* ESPS uses the *fea_type* field to indicate the FEA subtype. Currently defined subtypes in fea.h include FEA_SD (for sampled data), FEA_ANA (for speech analysis), FEA_STAT (for statistics), FEA_VQ (for vector quantization), and FEA_SPEC (for spectral records). An ASCII array of strings (char *fea_file_type[]) for these defined constants, suitable for use with *lin_search* (3−ESPS, is defined in the ESPS library. The definition is included automatically by means of *<esps/esps.h>*.

segment_labeled

If the value of this flag is YES, three fields are guaranteed to be defined: a coded field of size 1 named "source_file" and two long-integer fields of size 1 named "segment_start" and "segment_length". For each record these give the name of an ESPS file and the beginning record number and number of records of a segment in that file to which the feature record refers. If the flag value is NO, these fields need not be defined and in fact should not be defined; the three field names should be treated as reserved. The flag is set by the function *set_seg_lab*(3−ESPSu), and the programmer should not set it directly. A value of YES for *segment_labeled* is incompatible with a value of YES for *tag* in the common part of the header.

field_count

This is the number of distinct fields defined in the header. The items *names*, *sizes*, *ranks*, *dimens*, *types*, *starts*, and *enums* each point to the first element of an array of length *field_count* or (in the case of *names) field_count* + 1. Each element of each of these arrays refers to the field named by the corresponding element of *names.*

field_order

Normally, the data record in ESPS feature files is organized by data type. All of the doubles are written first, then all of the floats, then the longs, then the shorts, and then the character (byte) data. If this field is YES, then the data is written to (and read from) the disk file in the order that

the feature file fields were created by calling *add_fea_fld*. This feature is intended to be used by applications that must read or create files in an externally imposed format. The normal ESPS convention is more efficient. The default value for this field is NO.

names This points to the beginning of an array of strings that contains the field names and a terminating null string. The array is suitable as an argument of *lin_search2*(3−ESPSu). (Note that the difference between *lin_search* and *lin_search2* is that *lin_search* does a case insensitive compare.)

sizes This points to the beginning of an array of long integers. Each integer gives the number of items in the corresponding field.

ranks This points to the beginning of an array of short integers. Each integer gives the number of dimensions of the corresponding field (0 for a scalar, 1 for a vector, 2 for a matrix, etc.). If the rank of a field is 0, its size must be 1. This item and *dimens* may be NULL if there are no fields of 2 or more dimensions, and if there is no need to distinguish a scalar from a vector of length 1.

dimens This item points to the beginning of an array of pointers. Each of these pointers points to the beginning of a long-integer array that gives the dimensions of the corresponding field. The length of that integer array is the number of dimensions given in the array that *ranks* points to. The product of the dimensions must equal the field size given in the array that *sizes* points to. This item may be NULL if *ranks* is NULL or contains no entries greater than 1.

types This points to the beginning of an array of short integers. Each integer is a code that indicates the type of the items in the corresponding field. The allowed values for the codes are the integer constants DOUBLE, FLOAT, LONG, SHORT, BYTE, DOUBLE_CPLX, FLOAT_CPLX, LONG_CPLX, SHORT_CPLX, BYTE_CPLX, BIT, CHAR, and CODED, which are defined in <*esps/esps.h*>. The following table shows the C data type that corresponds to each code.

code	type
DOUBLE	double
FLOAT	float
LONG	long
SHORT	short
BYTE	signed char
DOUBLE_CPLX	double_cplx
FLOAT_CPLX	float_cplx
LONG_CPLX	long_cplx
SHORT_CPLX	short_cplx
BYTE_CPLX	byte_cplx
BIT	char
CHAR	char
CODED	short

The type codes DOUBLE, FLOAT, LONG, SHORT, and CHAR stand for the C types that the names suggest. Type codes DOUBLE_CPLX, FLOAT_CPLX, LONG_CPLX, SHORT_CPLX, and BYTE_CPLX stand for the complex types corresponding to the five floating and integer types. Typedefs for double_cplx, float_cplx, etc. are in the include file *esps/esps.h*. Type codes BYTE and CHAR stand for types of the same size, but BYTE is used to store byte-size integer data (signed), while CHAR is used for character data. The type code BIT stands for the ‘‘bit’’ data type. When available, this will occupy the same space in memory as BYTE or CHAR but will be packed into single bits in the external file. Type codes SHORT and CODED stand for types of the same size, but SHORT is used for arithmetic (signed integer) data, while CODED data consists of arbitrary codes that each designate one of a set of strings defined in the header. (See *enums* below).

enums This item points to the beginning of an array of pointers. Each of these pointers is NULL unless the corresponding type is CODED, and then it indicates the possible values for each item of the corresponding field. More specifically, the pointer, if not NULL, points to the beginning of a null-

terminated array of strings suitable as an argument of *lin_search2*(3−ESPSu); these strings are the possible values. Functions are available to find the code corresponding to a given string and vice versa; see *fea_encode*(3−ESPSu), *fea_decode*(3−ESPSu).

starts This points to the beginning of an array of longs. Each element gives the starting point of the data for the corresponding field relative to the pointer of the correct type in the data record. (Type CODED is treated as a SHORT).

derived This points to the beginning of an array of shorts, one for each field. A non-zero value means that the corresponding field was "derived" from another FEA file. That is, each element in the field corresponds to some particular element in another FEA file with a different field structure (see *srcfields*).

srcfields

This item points to the beginning of an array of character pointers. Each of these pointers is NULL unless the corresponding field is derived (see *derived*), in which case it points the beginning of a null-terminated array of strings. Each string has the form

<fieldname> [<element_range>]

where <fieldname> is a field name (usually not a field in the current FEA file), and where <element_range> is a list of elements in a form suitable for *grange_switch* (3−ESPS). The total number of elements described in this way must equal the size of the corresponding field. For example, suppose that a FEA file contains a derived field named *svector* of size 5. The contents of the corresponding array of strings in *srcfields* might be as follows: "raw_power[0]", "spec_param[1,3:5]". This is interpreted to mean that the five elements of *svector* were derived from elements in the *raw_power* and *spec_param* fields of some other FEA file − in particular, the 5 elements of *svector* correspond to *raw_power*[0], *spec_param*[1], *spec_param*[3], *spec_param*[4], and *spec_param*[5] (see *set_fea_deriv* (3−ESPSu)).

spares There are FEA_SPARES spare shorts.

ndouble, ndcplx, . . . , nbcplx

These give the total number of scalar items of each of the types DOUBLE, DOUBLE_CPLX, . . . , BYTE_CPLX in a record. The total for type CODED is included in *nshort* along with the total for SHORT, and the totals for types BIT and CHAR are included in *nbyte* along with the total for BYTE. Some of these items have the same names as items in the common part of the header, but their values may be different. For example *ndouble* in the common part of the header includes a count of 2 doubles for every double_cplx item in a FEA record and so is equal to *ndouble*+ 2**ndcplx* in terms of these members of the FEA-specific part of the header.

The data follows the header. The default data format in the file is that suggested by the following pseudo-C structure declaration.

```
struct fea_data {
    long    tag;                /* position tag */
    double  d_data[ndouble];    /* double data */
    float   f_data[nfloat];     /* float data */
    long    l_data[nlong];      /* long data */
    short   s_data[nshort];     /* short and coded data */
    char    b_data[char];       /* byte, char, and bit data */
};
```

The variables *ndouble, nfloat, nlong, nshort,* and *nchar* here refer to the items in the common part of the header. They cannot actually occur in a C declaration, but are used by the FEA support routines. An alternative external format is used if the item *field_order* has the value YES−−see *field_order* above.

In memory, the data is held in a structure like the one below, which is defined in <*esps/fea.h*>. The variables *ndouble, . . . , nbyte, . . . , nbcplx* here refer to the items in the FEA-specific part of the header. Again, these items do not actually occur in a C declaration; however, a function is available to allocate memory for

this data structure, based on the values in a particular header. See *allo_fea_rec*(3−ESPSu).

```
struct fea_data {
   long        tag;              /* position tag */
   double      d_data[ndouble];  /* double record items */
   float       f_data[nfloat];   /* float record items */
   long        l_data[nlong];    /* long record items */
   short       s_data[nshort];   /* short and coded record items */
   char        b_data[nbyte];    /* byte, char, and bit record items */
   double_cplx dc_data[ndcplx];  /* double complex record items */
   float_cplx  fc_data[nfcplx];  /* float complex record items */
   long_cplx   lc_data[nlcplx];  /* long complex record items */
   short_cplx  sc_data[nscplx];  /* short complex record items */
   byte_cplx   bc_data[nbcplx];  /* byte complex record items */
};
```

A feature file may have a position tag in each record. This tag refers to records in the file named in the header field *common.refer*. In addition there is space for *ndcplx* pairs of doubles, *ndouble* doubles, *nfcplx* pairs of floats, *nfloat* floats, *nlcplx* pairs of longs, *nlong* longs, *nscplx* pairs of shorts, *nshort* shorts, *nbcplx* pairs of bytes, and *nbyte* bytes, (where these are values from the FEA-specific part of the ESPS file header).

Space in *d_data* for fields of type DOUBLE is allocated in the order of occurrence of the fields' names in *names*. Space in the other data arrays in a record is allocated similarly. However, programs generally need not and should not depend on this information about record format. Functions are available to get a pointer to the beginning of the storage in a given record for the field with a given name; see *get_fea_ptr*(3−ESPSu).

EXAMPLES

Assume declarations

```
struct header   *hd;
struct fea_data *rec;
int             siz, rnk, *dim;
double          *pd;
short           *pe;
```

Suppose *hd* and *rec* have properly initialized, for example by *new_header*(3−ESPSu) or *read_header*(3−ESPSu) and by *allo_fea_rec*(3−ESPSu). Then the statement

```
pd = (double *)get_fea_ptr(rec, "energy", hd);
```

will assign to *pd* a pointer to the first double in the storage in *rec* for the field named "energy", provided that the field exists. Now a statement like

```
*pd = 3.7
```

will store a value into the field in *rec*. If the field has several elements, a subscripted variable like *pd[3]* can be used instead of **pd*. If *rec* already contains data, for example as a result of calling *get_fea_rec*(3−ESPSu), the data can be accessed by using **pd* or a subscripted *pd* in an expression. The statement

```
pe = (short *)get_fea_ptr(rec, "voicing", hd);
```

will assign to *pe* a pointer to a short integer in the storage in *rec* that holds a code for a value in the field named "voicing". A statement like

```
*pe = fea_encode("voiced", "voicing", hd);
```

will assign the code for the string "voiced" to the short integer, and the expression

```
fea_decode(*pe, "voicing", hd);
```

will get the string value corresponding to the code that is there.

SEE ALSO

allo_fea_rec(3−ESPSu), get_fea_rec(3−ESPSu), put_fea_rec(3−ESPSu), add_fea_fld(3−ESPSu), set_fea_deriv(3−ESPSu), get_fea_deriv(3−ESPSu), set_seg_lab(3−ESPSu), get_fea_ptr(3−ESPSu), fea_encode(3−ESPSu), fea_decode(3−ESPSu), new_header(3−ESPSu) read_header(3−ESPSu), write_header(3−ESPSu), lin_search(3−ESPSu), ESPS(5−ESPS)

FUTURE CHANGES

Make BIT type available. Support for packed bit fields in memory. Define additional values for the *type* field in the header.

AUTHOR

Manual page by Rodney Johnson. Incorporates suggestions by Joe Buck, Alan Parker, and John Shore. Implementation by Alan Parker.