DSPACK - Calling Sequences

C binding
(Version 1.27)

Ryszard Zybert
School of Physics and Space Research
University of Birmingham
(R.Zybert@bham.ac.uk)

19^{th} January 1995

Table of Contents

Chapter 1: C calling sequences

1.1 Introduction

All C calling sequences are defined as macros in file **dsc.h** in $DSPACK_HOME/inc. For ANSI C a prototype file **dsc_prot.h** is provided as well. All C calls which correspond to FORTRAN subroutines are changed into int functions returning 1 on success and 0 on failure. All other calls return 0 on failure and non-zero on success.

Many calls have a common sequence of arguments:

`char *name`	- Object name
`int *ivdsn`	- Object handle (if 0, correct handle is returned)
`...`	
`int *ierr`	- Returned error flag - should be 0

name should be NULL terminated.

ivdsn should be declared static, initialized to 0 and never modified. There should be one **ivdsn** for each **name**. This procedure speeds up access to objects by avoiding name search.

If ivdsn$>$ 0 on input, the name is checked against the name of the object pointed to by ivdsn - if it does not agree, ivdsn is corrected.

If name is not found, ivdsn is set to a negative of the total number of objects defined at that time. In subsequent calls the search for name is only performed if new objects have been defined since the last failure.

If name starts from ”-” then it is assummed that ivdsn is correct and nothing is checked.

1.2 DSPACK Calls

1.2.1 Initialization

int ds_carg(argv)

`char **argv`	pointer to argument list

(Only if the main program is in C)

ds_carg passes command line argument list to DSPACK. It should be called before DSPACK is initialized. Always returns 1.

int dsinit(mode, name, lun, ds_size, sect2_size, ierr)

`int mode`	0 - LOCAL, 1 - CLIENT
`char *name`	Server and client name separated by a ”/” (eg. ”myserver/myprogram”). Server name may be ommitted.
`int lun`	Fortran unit number for error output - usually 6.
`int ds_size`	Initial size of DSPACK memory in words - 0 for minimum size.
`int sect2_size`	Initial size of DSPACK section 2 memory in words - 0 for minimum size.
`int *ierr`	Returns error flag - 0 means OK.

Returns 1 for success and 0 for failure. Initializes DSPACK. Should be called once - other calls are ignored.

If lun=0, all diagnostic messages are suppressed.

Example

```
#include "dsc_prot.h"
#include "dsc.h"
void main(argc,argv)
    int argc;
    char **argv;
{
  int ierr;

  ds_carg(argv);

  if ( !dsinit(0,"test",6,0,0,&ierr) )
    {
     printf("DSINIT error: %i\n",ierr);
     exit(100);
    }
```

1.2.2 Access to data

1.2.2.1 Getting data

int dsget(name, iv, mxent, ncopy, idata, ierr)

`char *name`	Object name
`int *ivdsn`	Object handle (if 0, correct handle is returned)
`int mxent`	maximum number of entries to copy
`int *ncopy`	returns number of entries copied
`void *idata`	pointer to memory to accept data
`int *ierr`	Returned error flag - should be 0

Copies up to mxent entries of an object into user's memory starting with entry 1. If the number of existing entries is greater than mxent, mxent entries are copied and ierr=104.

void *idsget_ds(name, ivdsn, ngot)

void *idsget_ds_nc(name, ivdsn, ngot)

`char *name`	Object name
`int *ivdsn`	Object handle (if 0, correct handle is returned)
`int *ngot`	returns number of existing entries

Returns the pointer to the first entry of an object or 0 on failure. **idsget_ds_nc** turns off class handling for this call. Entries are contiguous in DSPACK memory. Consequently the pointer should be treated as a pointer to an array of structures - other entries may be reached by indexing the array or by appropriate casting and incrementing the pointer.

int dsgitt(name, ivdsn, idat, ient, tx, ierr)

`char *name`	Object name
`int *ivdsn`	Object handle (if 0, correct handle is returned)
`int idat`	item number
`int ient`	entry number

`char *tx`	pointer to user's memory
`int *ierr`	Returned error flag - should be 0

Returns item value as text. **tx** should be long enough to take a value of the item in printable form.

int ids_wait(name, ivdsn, iflag)

`char *name`	Object name
`int *ivdsn`	Object handle (if 0, correct handle is returned)
`int iflag`	mode flag 0- function returns when the object has data either immediately or after update 1- function returns after next update

In client/server mode this function waits for the object to be filled or updated - depending on **iflag**. It returns the number of entries. In local mode the function returns immediately.

void ∗idsget(name, iv, ient, isize, ngot, mxent, ncopy, idata, ierr)

`char *name`	Object name
`int *iv`	Call ID. It should be declared static and **must** be preset to 0 before the first call. It is set to an identifier which allows to access the object without parsing the name again.
`int ient`	Entry number (not used if entry is specified within the name)
`int *size`	Returns size of single entry of the root object in bytes
`int *ngot`	Returns the number of entries in root object
`int mxent`	Maximum number of entries to copy into idata
`int *ncopy`	Returns the number of entries copied into idata
`void *idata`	Pointer to user's memory to return data
`int *ierr`	Returned error flag - should be 0

Access to individual items of an object.

This routine returns a pointer to a fully specified item of an object. Additionally data may be copied into user's array if mxent$>$ 0. Entry number may be either specified as a part of the name or in argument ient. **isize** returns the size in bytes of the root object.

If mxent$>$ 0 then up to mxent entries are copied into idata by taking the specified item from each entry of the root object.

idsget is more complicated to use and generally slower than **idsget_ds** so it should only be used when needed.

NOTE: iv in idsget is not the object handle and it must not be mixed with ivdsn in other similar calls. Returned iv points to a descriptor which contains the object handle as well as the offset to the requested item and the entry number. If iv is non-zero on input the name is not parsed or checked at all.

Examples

```
typedef point {
  float x;
  float y;
  float z;
};
struct point sp[1000];
struct point *pnt_p;
float  x[1000];
```

```
float  *x_p;
static int p_h=0;
static int x_h=0;
int isize,ngot,ncopy,ierr;

pnt_p = (struct point *)idsget("track.start",&p_h,1,&isize,&ngot,
                                    0,0,0,&ierr);

/* If track.start is a structure of type  point within track
   structure then this call returns the pointer to the start of
   the first track.   isize is the size of track in bytes - so
   it is the distance in bytes to the start structure of the next
   track */

idsget("track.start",&p_h,1,&isize,&ngot,1000,&ngot,sp,&ierr);

/* This call copies up to 1000 start points - one per track - into
   sp */

x_p = (float *)idsget("track(1).start.x",&x_h,0,&isize,&ngot,
                                    0,0,0,&ierr);
/* If track.start is a structure of type  point within
   track structure then this call returns the pointer to the
   x coordinate of the start of the first track. Note - ient
   is ignored here as the name contains the entry number */

idsget("track.start.x",&x_h,1,&isize,&ngot,1000,&ngot,x,&ierr);
/* This call copies up to 1000 start.x values - one per track
   - into x */
```

1.2.2.2 Storing data

int dsput(name, ivdsn, nent, idata, ierr)

`char *name`	Object name
`int *ivdsn`	Object handle (if 0, correct handle is returned)
`int nent`	number of entries to store
`void *idata`	pointer to memory containing data
`int *ierr`	Returned error flag - should be 0

Copies nent entries from user's memory into an object. New data replace any existing data stored previously. **nent** becomes new number of entries.

void *idsput_ds(name,ivdsn,nent)

`char *name`	Object name
`int *ivdsn`	Object handle (if 0, correct handle is returned)
`int nent`	number of entries to store

Allocates memory for **nent** entries of an object and returns the pointer to the first entry. Existing entries, if any, are lost.

void *ds_putt(text, ierr)

`char *text`	Assignment expression

`int *ierr`	Returned error flag - should be 0

Parses assignment expression and executes it.

Assignment should be in the form **name = value** - where **name** is a fully qualified name of an object element.

value may contain reference to other objects in the form **{name}** or references to environment variables in the form **$name**.

If the specified entry does not exist, the memory is allocated for it (and for all preceeding entries).

Examples

```
ds_putt("xx(3).k = 3",&ierr);
ds_putt("xx(1).name = $USER",&ierr);
ds_putt("yy(2).k = {xx(3).k}",&ierr);
```

void *ids_add_entry(name, ivdsn, nent, nstep, ibase, ierr)

`char *name`	Object name
`int *ivdsn`	Object handle (if 0, correct handle is returned)
`int nent`	number of entries to add
`int nstep`	minimum number of entries to allocate when needed
`void **ibase`	returns the pointer to entry 1.
`int *ierr`	Returned error flag - should be 0

Adds **nent** entries to an object. If enough memory has been allocated for this object already then the new entries use that memory, otherwise the object is extended (and moved if necessary). If the object needs to be resized the number of extra entries allocated is nent or nstep, whichever is greater. Setting nstep to expected final number of entries saves cpu time as the object does not have to be relocated. If relocation is necessary all pointers in all DSPACK objects which point to this object are corrected. This may be quite expensive - therefore nstep should be chosen carefully.

The function returns the pointer to the first added entry, the pointer to the first entry is returned in **ibase**.

When the object is ready **ds_update** should be called to inform DSPACK.

int dsbook(name, ivdsn, nent, ierr)

`char *name`	Object name
`int *ivdsn`	Object handle (if 0, correct handle is returned)
`int nent`	number of entries to allocate
`int *ierr`	Returned error flag - should be 0

Allocates memory for nent entries of object **name**. Data may be stored by calling **dsadde** or **idsadd**. If the object contains data the existing entries are preserved (up to nent), however if the re-sizing of the object involves relocation, no attempt is made to correct existing user's pointers. If there are valid pointers in other DSPACK objects pointing to object **name** - then **idsrebook** should be used instead.

void *idsrebook(name, ivdsn, nent, ierr)

`char *name`	Object name
`int *ivdsn`	Object handle (if 0, correct handle is returned)

`int nent`	New number of entries.
`int *ierr`	Returned error flag - should be 0

Re-allocates memory for the object **name**. If the object exists, then it is re-sized. If this involves re-location, all existing pointers in DSPACK objects are corrected.
Returns the new pointer to the first entry.

void *idsadd(name, ivdsn, ient, ngot, isize, nentot, ierr)

`char *name`	Object name
`int *ivdsn`	Object handle (if 0, correct handle is returned)
`int ient`	Number of entries to add.
`int *ngot`	Number of entries added.
`int *isize`	Size of the root object in bytes.
`int *nentot`	Total number of entries after this call.
`int *ierr`	Returned error flag - should be 0

Adds ient entries to an object - in memory allocated previously by **dsbook** or **idsrebook**.
Returns the pointer to the first added entry.

int dsadde(name, ivdsn, nenti, nentot, data, ierr)

`char *name`	Object name
`int *ivdsn`	Object handle (if 0, correct handle is returned)
`int nenti`	Number of entries to add.
`int *nentot`	Total number of entries after this call.
`void *data`	Data.
`int *ierr`	Returned error flag - should be 0

Adds nenti entries to object **name**. New entries are copied from user's memory **data**. Entries have to be allocated previously by **dsbook** or **idsrebook** call.

int dsrepl(name, ivdsn, ient, nent, data, ierr)

`char *name`	Object name
`int *ivdsn`	Object handle (if 0, correct handle is returned)
`int ient`	First entry to replace.
`int nent`	Number of entries to replace.
`void *data`	Data.
`int *ierr`	Returned error flag - should be 0

Replaces nent entries of object **name** starting from entry **ient** with **data**. All replaced entries must exist.

int dsset(name, ivdsn, ient, nent, ix, ierr)

`char *name`	Object name

`int *ivdsn`	Object handle (if 0, correct handle is returned)
`int ient`	First entry to set.
`int nent`	Number of entries to set.
`int ix`	Constant.
`int *ierr`	Returned error flag - should be 0

Sets memory of nent entries of an object, starting with entry **ient** to a constant ix.

1.2.3 Operations on Objects

int ds_update(name, ivdsn, ierr)

`char *name`	Object name
`int *ivdsn`	Object handle (if 0, correct handle is returned)
`int *ierr`	Returned error flag - should be 0

Informs DSPACK that the object is ready. In client/server mode the server will wake up all clients waiting for that object.
In all modes the unused but allocated entries are removed.

int ds_update_ns(name, ivdsn, ierr)

`char *name`	Object name
`int *ivdsn`	Object handle (if 0, correct handle is returned)
`int *ierr`	Returned error flag - should be 0

Informs DSPACK that the object is ready. In client/server mode the server will wake up all clients waiting for that object.
The allocated memory is left untouched.

int dsnclr(section)

`int section`	Section number

Clears DSPACK section. All objects which belong to this section are removed and all memory allocated for this section (except the initial allocation by dsinit) is returned to the system.
Section 2 cannot be cleared.

int dsclr(name, ivdsn, nn, ierr)

`char *name`	Object name
`int *ivdsn`	Object handle (if 0, correct handle is returned)
`int nn`	Number of entries the object should have after this call
`int *ierr`	Returned error flag - should be 0

This call clears the memory used by an object to 0. At the same time the number of entries of the object may be changed.

int dsdrop(name, ivdsn, ierr)

`char *name`	Object name
`int *ivdsn`	Object handle (if 0, correct handle is returned)
`int *ierr`	Returned error flag - should be 0

Delete object **name**. The object remains defined but all allocated entries are deleted.

int dshide(name, ivdsn, lpoint, nent, ierr)

`char *name`	Object name
`int *ivdsn`	Object handle (if 0, correct handle is returned)
`int *lpoint`	Direct link to the object
`int *nent`	Number of entries
`int *ierr`	Returned error flag - should be 0

Make object invisible to DSPACK.

All existing pointers remain valid and the memory used by the object is preserved but all DSPACK calls referring to that object return nothing.

This allows a new version of the object to be stored while the old version is preserved.

lpoint should be kept by the calling program - it will be needed to restore or delete the hidden object.

int dsrest(ivdsn, lpoint, chopt, ierr)

`char *name`	Object name
`int *ivdsn`	Object handle (if 0, correct handle is returned)
`int *lpoint`	Direct link to the object as returned by **dshide**
`char *chopt`	Option deciding what to do if a new version of the object exists:
	”f”- force - delete the new version
	”s”- swap - hide the new version - lpoint returns new value
	” ”- preserve - if the new version exists, do nothing.
`int *ierr`	Returned error flag - should be 0

Restores the object hidden by **dshide**. If a new version of the object has been stored in the mean-time the action depends on **chopt**

int dssel(name, ivdsn, maxent, nent, flist, ierr)

`char *name`	Object name
`int *ivdsn`	Object handle (if 0, correct handle is returned)
`int maxent`	Number of entries to test
`int *nent`	Final number of entries
`int *flist`	Selection array
`int *ierr`	Returned error flag - should be 0

Selects entries of the object. Selection is made on the basis of user's supplied array of size at least maxent. Each element of the array corresponds to one entry of the object. All entries corresponding to elements of flist which are zero are removed from the object. The data are compressed so that the remaining entries are contiguous. No pointer handling is attempted at present.

int ds_map_mem(name, ivdsn, iaddr, ierr)

`char *name`	Object name
`int *ivdsn`	Object handle (if 0, correct handle is returned)
`void *iaddr`	New address of the object
`int *ierr`	Returned error flag - should be 0

Map object **name** into memory pointer to by *iaddr. The target address must be preceeded by a DSPACK object header in order for the object to be recognized by DSPACK. (See ids_create_priv).

int ds_map_object(name1, iv1, name2, iv2, ierr)

`char *name1`	First object name
`int *iv1`	Object handle (if 0, correct handle is returned)
`char *name2`	Second object name
`int *iv2`	Object handle (if 0, correct handle is returned)
`int *ierr`	Returned error flag - should be 0

Map object **name1** into memory allocated for object **name2**. Mapping is temporary, the pointer to object **name1** does not change when object **name2** is relocated or deleted. Also the number of entries of object 1 does not change if that of object 2 does.

int ds_mv_object(name1, iv1, name2, iv2, ierr)

`char *name1`	First object name
`int *iv1`	Object handle (if 0, correct handle is returned)
`char *name2`	Second object name
`int *iv2`	Object handle (if 0, correct handle is returned)
`int *ierr`	Returned error flag - should be 0

Copy object **name1** into object **name2**. If object 2 exists then it's pointer is reset to point to data of object 1 and the number of entries is reset to the corresponding value (which may or may not be equal to the number of entries of object 1 - the objects do not have to be identical). Object 1 vanishes.

If object 2 is not defined it will be defined as identical to object 1 - this effectively corresponds to a 'rename'.

If object 2 is defined and it is in a different section than object 1, then the data are copied into a new object 2 and object 1 is deleted.

int ids_pnt2h(iaddr, ient)

`void *iaddr`	Pointer to DSPACK object
`int *ient`	Corresponding entry number

Converts a pointer to object handle. **ient** returns the entry number.

void ∗ids_create_priv(name, nent, descr, ierr)

`char *name`	Object name
`int nent`	Number of entries
`struct ds_ext_obj *descr`	Returned object descriptor
`int *ierr`	Returned error flag - should be 0

Allocates memory and creates 'private' object with DSPACK header. **name** must refer to a defined DSPACK object. The user has to provide descr structure (defined as ds_ext_obj in $DSPACK_HOME/inc/dspack_z.h). The information returned in descr may be subsequently used to delete the object or to release it as a DSPACK object.

The pointer to the new object is returned.

In client/server mode the object is invisible to other clients until ids_release is called.

int ds_delete_priv(descr,ierr)

`struct ds_ext_obj *descr`	Object descriptor from ids_create_priv
`int *ierr`	Returned error flag - should be 0

Delete private object. The memory is returned to the system

int ids_release(descr)

`struct ds_ext_obj *descr`	Object descriptor from ids_create_priv

Make a private object visible via DSPACK calls. If successful the call returns object handle, otherwise it returns 0.

void ∗ids_get_external(server, name, descr, ierr)

`char *server`	Server name
`char *name`	Object name
`struct ds_ext_obj *descr`	Returned object descriptor
`int *ierr`	Returned error flag - should be 0

Gets an object from another server.

The pointer to the object is returned.

The user has to provide descr structure (defined as ds_ext_obj in $DSPACK_HOME/inc/dspack_z.h). It may be subsequently used in ids_release.

The object becomes available to the calling program via returned pointer. In order to make it available via DSPACK calls (and to make it accessible by other clients) ids_release has to be called.

If the object contains no pointers, then DSPACK simply attaches to the shared memory segment of another server. However - the local address of the memory segment may be different from the original address in the owner's program - therefore, if the object contains pointers, then a copy of the object is made before it is accessible as a DSPACK object.

Until the object is released all the pointers are represented by entry numbers of corresponding objects. The user must not store any pointers in an external object until ids_release is called.

1.2.4 I/O

int fsopin(input_id ,file, chopt, ierr)

`int input_id`	DSPACK input unit
`char *file`	File name
`char *chopt`	Options - not used at present, use " "
`int *ierr`	Returned error flag - should be 0

Opens DSPACK input file - DSPACK unit may be between 0 and 9. Input and output units are independent.

int fsopou(output_id ,file, chopt, ierr)

`int output_id`	DSPACK output unit
`char *file`	File name
`char *chopt`	Options - not used at present, use " "
`int *ierr`	Returned error flag - should be 0

Opens DSPACK output file - DSPACK unit may be between 0 and 9. Input and output units are independent.

int fsmark(output_id, name, ntot, ierr)

`int output_id`	DSPACK output unit
`char *name`	Object name or regular expression.
`int *ntot`	Number of objects marked after this call
`int *ierr`	Returned error flag - should be 0

Marks objects for output on unit **output_id**. If a complete object name is specified, this object is marked, otherwise name is treated as a regular expression and all objects which agree with it are marked.

If name is specified as "*" then all object not starting with a *period* are marked.

int fsumrk(output_id, name, ntot, ierr)

`int output_id`	DSPACK output unit
`char *name`	Object name or regular expression.
`int *ntot`	Number of objects marked after this call
`int *ierr`	Returned error flag - should be 0

Removes an object from the output list.

int fsoutv(output_id, section, nout, ierr)

`int output_id`	DSPACK output unit
`int section`	Section number
`int *nout`	Number of objects marked after this call
`int *ierr`	Returned error flag - should be 0

Write all marked objects in a given section to output. If the object list has changed since the last call, all definitions are written out as well.

Pointers in objects are not followed, but the definitions of all objects pointed to from written objects are written into the definition block.

int fdsin(input_id, ktyp, ierr)

`int input_id`	DSPACK input unit
`int *ktyp`	Record type: 0- Definition block 1- Variables 2- Constants n- Section n
`int *ierr`	Returned error flag - should be 0

Read one logical record from a DSPACK file. **ktyp** returns the record type. **ierr = 2** means End-Of-File.

All objects from input are read unless the *lock* flag is set for them - see **ds_lock**.

If an object read from input exist already, it is replaced. Existing objects may be protected by hiding them with **dshide** or by using **ds_lock**.

If an object is defined before opening the input file, the definition has to agree with the definition in the input file. If the object in the file is in different section than the one in the program the definition in the program is used.

The mechanism of mapping objects which are differently defined in the program and in the file is in preparation.

int fsendo(output_id)

`int output_id`	DSPACK output unit

Close output unit. This call is obligatory in order to flush all output buffers, write trailing records, file-marks on tape etc.

int fsendi(input_id)

`int input_id`	DSPACK input unit

Close input unit.

int fsclr(output_id, ierr)

`int output_id`	DSPACK output unit
`int *ierr`	Returned error flag - should be 0

Clear output list - unmarks all objects marked for given output unit.

int ds_io_lst(input_id, isect, label, nn, ierr)

`int input_id`	DSPACK input unit
`int isect`	Section - 0 means all sections
`char *label`	Returns object name

`int *nn`	Count - should be set to 0 on the first call
`int *ierr`	Returned error flag - 0 - OK, 1 - no more

Returns names of objects found on input. Should be called repeatedly until ierr=1 is returned. Each call returns one object name

int ds_oo_lst(output_id, isect, label, nn, ierr)

`int output_id`	DSPACK output unit
`int isect`	Section - 0 means all sections
`char *label`	Returns object name
`int *nn`	Count - should be set to 0 on the first call
`int *ierr`	Returned error flag - 0 - OK, 1 - no more

Returns names of objects marked for output. Should be called repeatedly until ierr=1 is returned. Each call returns one object name.

int ds_lock(name, ivdsn, ierr)

`char *name`	Object name
`int *ivdsn`	Object handle (if 0, correct handle is returned)
`int *ierr`	Returned error flag - should be 0

Locks an object against input. The object **name** will be ignored on input.

int ds_unlock(name, ivdsn, ierr)

`char *name`	Object name
`int *ivdsn`	Object handle (if 0, correct handle is returned)
`int *ierr`	Returned error flag - should be 0

Unlocks an object - turns off the effect of **ds_lock**.

int fscopy(input_id, output_id, ntot)

`int input_id`	DSPACK input unit
`int output_id`	DSPACK output unit
`int *ntot`	Number of objects found

Marks all objects found on input **input_id**. The list of input objects is copied into the list of output objects. No I/O operation is performed.

int fsshve(lun)

`int lun`	Fortran I/O unit - 0 or 6 for standard output

Writes (on Fortran unit *lun*) DSPACK version and date of the program which wrote the file being read. The call refers to the last file accessed.

int fsvers(vds, tds, idate, itime)

`float *vds`	DSPACK version
`float *tds`	DSPACK date - YYMMDD.0
`int *idate`	File creation date - YYMMDD
`int *itime`	File creation time - HHMMSS

Returns the information about DSPACK file. The calls refers to the last file accessed.

int ifs_iunit(ierr)

`int *ierr`	Returned error flag - should be 0

Returns the next free DSPACK input unit. (-1 on failure)

int ifs_ounit(ierr)

`int *ierr`	Returned error flag - should be 0

Returns the next free DSPACK output unit. (-1 on failure)

1.2.5 Pointers

int ds_toind(name, ivdsn, ierr)

`char *name`	Object name
`int *ivdsn`	Object handle (if 0, correct handle is returned)
`int *ierr`	Returned error flag - should be 0

Converts all pointers stored in a DSPACK object to indices. Each pointer is replaced by the corresponding entry number in the target object. The conversion is performed recursively for all objects which can be reached by pointers from the current object.

int ds_topnt(name, ivdsn, ierr)

`char *name`	Object name
`int *ivdsn`	Object handle (if 0, correct handle is returned)
`int *ierr`	Returned error flag - should be 0

Converts all indices stored in a DSPACK object to pointers. The conversion is performed recursively for all objects which can be reached by pointers from the current object.

int ids_pnt_conv(new_mode)

`int new_mode`	New mode: 1 - pointers, 2 - indices

Sets *pointer/index* mode switch for all subsequent DSPACK calls. If an object contains pointers and the mode is set to indices - any call getting data will convert all pointers to indices and vice versa. Also all created object will expect indices to be stored by the user in place of pointers.

The function returns the value of the switch before the call.

1.2.6 Resources

Resources are stored as a link list of unspecified objects attached to the definition of an object. The user is free to attach his private resource to an object. All resources are written into the file header when the object is written to output.

Each resource has a 4 character identifier. More than one resource with a given identifier may be attached to an object.

Resources should not contain pointers - the user is fully responsible for the information in the resource to be valid when the resource is written to output. DSPACK performs no conversion of resource data.

Resources become a part of the object definition. They should not be changed after the header has been written to output.

As a convention - DSPACK's own resources have identifiers in upper case. User defined resources should have identifiers in lower or mixed case.

int ids_add_res(name, ivdsn, type, nw, ibuf, ierr)

`char *name`	Object name
`int *ivdsn`	Object handle (if 0, correct handle is returned)
`char *type`	Resource ID
`int nw`	Resource length in words (if negative, allocate only)
`void *ibuf`	Resource data (ignored if nw is negative)
`int *ierr`	Returned error flag - should be 0

Adds a resource to an object. The pointer to the new resource is returned.

If $nw < 0$ the memory ($abs(nw)$ words) is allocated but no data are copied, If $nw > 0$ nw words are copied from *ibuf* into the new resource.

int ids_put_res(name, ivdsn, type, nw, ibuf, ierr)

`char *name`	Object name
`int *ivdsn`	Object handle (if 0, correct handle is returned)
`char *type`	Resource ID
`int nw`	Resource length in words (if negative, allocate only)
`void *ibuf`	Resource data (ignored if nw is negative)
`int *ierr`	Returned error flag - should be 0

Similar fo ids_add_res but if a resource of type *type* is found then it is replaced by the new resource.

int ids_get_res(name, ivdsn, type, nlev, nw, ierr)

`char *name`	Object name
`int *ivdsn`	Object handle (if 0, correct handle is returned)
`char *type`	Resource ID
`int *nlev`	Resource counter - should be zero on the first call
`int *nw`	Resource length in words
`int *ierr`	Returned error flag - should be 0

Returns the pointer to a resource. If more than one resource of given ID exist for object *name* the function should be called repeatedly. In any case nlev should be set to 0 before the first call and not changed between calls. The function will return given resources one by one and will return NULL if there is no more resources with given ID.

More than one resource ID may be specified in *type* - separated by commas. The effect is an **or** of different IDs - resources with ID which is on the list will be returned in the order in which they are stored.

int ids_del_res(name, ivdsn, type, ierr)

`char *name`	Object name
`int *ivdsn`	Object handle (if 0, correct handle is returned)
`char *type`	Resource ID
`int *ierr`	Returned error flag - should be 0

Deletes resource *type* from object *name*. Only the first resource found is deleted.

int ids_rpl_res(res, nw, ibuf, ierr)

`void *res`	Resource pointer
`int nw`	Resource length in words (if negative, allocate only)
`void *ibuf`	Resource data (ignored if nw is negative)
`int *ierr`	Returned error flag - should be 0

Replace resource - if the pointer to the resource is known - it may be used instead of ids_put_res to replace existing resource.

nw is the new resource length - this call makes sense only if the resource length changes - otherwise the data may be simply written into *res*.

int dsadco(name, ivdsn, text, ierr)

`char *name`	Object name
`int *ivdsn`	Object handle (if 0, correct handle is returned)
`char *text`	Comment text - 80 characters
`int *ierr`	Returned error flag - should be 0

Adds a line of comments - 80 characters - to the object definition.

int dsgtco(name, ivdsn, nskip, text, ierr)

`char *name`	Object name
`int *ivdsn`	Object handle (if 0, correct handle is returned)
`int *nlev`	Resource counter (set to 0 before the first call)
`char *text`	Comment text - 80 characters
`int *ierr`	Returned error flag - should be 0

Returns a line of comment for the object *name*. If more than one line of comment exist, it should be called repeatedly - for each comment line. In any case nlev should be set to 0 before the first call and not changed between calls. The function will return comment lines one by one and will return NULL if there is no more comment lines.

1.2.7 Defining objects

int dsdef(name, type, isect, ndat, labels, numv, ierr)

`char *name`	Object name
`char *type`	Object type **'I'**- integer **'R'**- real **'L'**- logical **'D'**- double precision **'B'**- bit string **'C*n'**- fixed character - n bytes per item (mod(n,4)=0) **'C'**- character - only single entry of any length **'S'**- structure
`int isect`	DSPACK section **1**- event section, short term **2**- constant section - long term **0**- no section - used to define templates. Objects with isect=0 cannot hold data - they may be used to define other objects.
`int ndat`	Number of items
`int *ivdsn`	Item names •- Elements are separated by spaces or character '|'. Separators must not be mixed - use either spaces or '|' in the whole string. •- There must be at least NITEM elements. •- Arrays (only one-dimensional) must be dimensioned as $name(number)$ •- Pointers must start with an $*$ •- If TYPE='S' there must be at least 2*NITEM elements. The last NITEM elements contain names of existing objects which specify types. Templates for simple types are defined at initialization: **.REAL4**- float **.INT4**- int **.DOUBLE8**- double **.CHAR4**- char[4] **...**
`int *ierr`	Returned error flag - should be 0

int dslike(newnam, oldnam, isec, ivdsn, ierr)

`char *name`	Object name
`char *name`	Template name
`int isec`	DSPACK section
`int *ivdsn`	Returns the new object handle
`int *ierr`	Returned error flag - should be 0

Defines object **newnam** using a template **oldnam**. **oldnam** may be the name of any existing object. The new object may be in different section than the old object.

int dsdefl(name, ivdsn, nlnk, labels, ierr)

`char *name`	Object name
`int *ivdsn`	Object handle (if 0, correct handle is returned)
`int nlnk`	Number of links
`char *labels`	List of links in form $< linkname >:< objectname >$ separated by commas or blanks
`int *ierr`	Returned error flag - should be 0

Links an object with other objects. All links for the object should be specified in one call. Each link has a name and connects the object to another existing objects. Link names may be used in refering to object. The links are followed on output.

1.2.8 Object info

int dschek(name, ivdsn, ierr)

`char *name`	Object name
`int *ivdsn`	Object handle (if 0, correct handle is returned)
`int *ierr`	Returned error flag - should be 0

Checks object handle against object name. If the handle is 0 or is positive and wrong - correct handle is returned. If object is not found $ivdsn < 0$ is returned. If $ivdsn < 0$ on input then the search is only performed if new objects have been defined since this ivdsn was returned. Returns 1 on success and 0 on failure.

int ids_handle(name)

`char *name`	Object name

Returns object handle or 0 if not found.

int dsivid(name, item, ivdsn, idat, ierr)

`char *name`	Object name
`char *item`	Item name
`int *ivdsn`	Returns object handle
`int *idat`	Returns item number
`int *ierr`	Returned error flag - should be 0

Returns the object handle and item number.

int ids_size(name, ivdsn)

`char *name`	Object name
`int *ivdsn`	Object handle (if 0, correct handle is returned)
`int *ierr`	Returned error flag - should be 0

Returns size in bytes of an entry of an object. If name refers to an item rather than the root object - the size of of the item is returned. In this case ivdsn is the object descriptor rather than a handle.

int ids_room(name, ivdsn, nent, ierr)

`char *name`	Object name
`int *ivdsn`	Object handle (if 0, correct handle is returned)
`int *nent`	Number of free entries.
`int *ierr`	Returned error flag - should be 0

Returns (in nent) the number of free entries (allocated but not filled) in an object.

int ids_root(name)

`char *name`	Object name

Returns number of characters of **name** which form the *root name*

int ids_nent(name, ivdsn)

`char *name`	Object name
`int *ivdsn`	Object handle (if 0, correct handle is returned)
`int *ierr`	Returned error flag - should be 0

Returns current number of entries in an object.

int ds_ainfo(name, isize, nelem, ierr)

`char *name`	Object name
`int *isize`	Element size in bytes
`int *nelem`	Number of elements
`int *ierr`	Returned error flag - should be 0

If **name** refers to an array - isize and nelem return the element size in bytes and the number of elements in the array.

int dsshow(name , ient, lun, ierr)

`char *name`	Object name
`int ient`	Entry number (0 means ALL)
`int lun`	Fortran unit number - usually 6
`int *ierr`	Returned error flag - should be 0

Displays formatted object data.

int dsshdf(name, ivdsn, lun, ierr)

`char *name`	Object name
`int *ivdsn`	Object handle (if 0, correct handle is returned)

`int lun`	Fortran unit number - usually 6
`int *ierr`	Returned error flag - should be 0

Displays object definition.

int dsshdc(name, ivdsn, icnt, lun, ierr)

`char *name`	Object name
`int *ivdsn`	Object handle (if 0, correct handle is returned)
`int *ierr`	Returned error flag - should be 0

int dsinpc(input_id, lun)

`char *name`	Object name
`int *ivdsn`	Object handle (if 0, correct handle is returned)
`int *ierr`	Returned error flag - should be 0

int ids_get_def(name)

`char *name`	Object name
`int *ivdsn`	Object handle (if 0, correct handle is returned)
`int *ierr`	Returned error flag - should be 0

int ids_get_alldef(nobj)

`char *name`	Object name
`int *ivdsn`	Object handle (if 0, correct handle is returned)
`int *ierr`	Returned error flag - should be 0

int dsallc(lun)

`char *name`	Object name
`int *ivdsn`	Object handle (if 0, correct handle is returned)
`int *ierr`	Returned error flag - should be 0

int dsallh(lun)

`char *name`	Object name
`int *ivdsn`	Object handle (if 0, correct handle is returned)
`int *ierr`	Returned error flag - should be 0

int dsouph(ioid, lun)

`char *name`	Object name

`int *ivdsn`	Object handle (if 0, correct handle is returned)
`int *ierr`	Returned error flag - should be 0

int dsfhdr(name, ivdsn, lun, ierr)

`char *name`	Object name
`int *ivdsn`	Object handle (if 0, correct handle is returned)
`int *ierr`	Returned error flag - should be 0

int dsglnk(name, ivdsn, n, naml, name1, ivdsn1, ntot, ierr)

`char *name`	Object name
`int *ivdsn`	Object handle (if 0, correct handle is returned)
`int *ierr`	Returned error flag - should be 0

int dsglnn(name, ivdsn, naml, name1, ivdsn1, ntot, nlnk, ierr)

`char *name`	Object name
`int *ivdsn`	Object handle (if 0, correct handle is returned)
`int *ierr`	Returned error flag - should be 0

int dsgtrg(name, trgnam, ivtarg, ierr)

`char *name`	Object name
`int *ivdsn`	Object handle (if 0, correct handle is returned)
`int *ierr`	Returned error flag - should be 0

int dssidf(name, lun, ierr)

`char *name`	Object name
`int *ivdsn`	Object handle (if 0, correct handle is returned)
`int *ierr`	Returned error flag - should be 0

int dshdrf(name, ivdsn, icnt, lun, ierr)

`char *name`	Object name
`int *ivdsn`	Object handle (if 0, correct handle is returned)
`int *ierr`	Returned error flag - should be 0

int ds_name(ivdsn, name, ierr)

`char *name`	Object name
`int *ivdsn`	Object handle (if 0, correct handle is returned)
`int *ierr`	Returned error flag - should be 0

1.2.9 Interactive

int ds_examine(name)

`char *name`	Object name
`int *ivdsn`	Object handle (if 0, correct handle is returned)
`int *ierr`	Returned error flag - should be 0

1.2.10 Error handling

int ds_error(name, text, ierr, lun)

`char *name`	Object name
`int *ivdsn`	Object handle (if 0, correct handle is returned)
`int *ierr`	Returned error flag - should be 0

int ds_print(text, lun)

`char *name`	Object name
`int *ivdsn`	Object handle (if 0, correct handle is returned)
`int *ierr`	Returned error flag - should be 0

1.2.11 Utilities

int dsexna(text)

`char *name`	Object name
`int *ivdsn`	Object handle (if 0, correct handle is returned)
`int *ierr`	Returned error flag - should be 0

int dsexnu(name, ierr)

`char *name`	Object name
`int *ivdsn`	Object handle (if 0, correct handle is returned)
`int *ierr`	Returned error flag - should be 0

int ldsexn(name,ierr)

`char *name`	Object name
`int *ivdsn`	Object handle (if 0, correct handle is returned)
`int *ierr`	Returned error flag - should be 0

int idslen(text)

`char *name`	Object name
`int *ivdsn`	Object handle (if 0, correct handle is returned)

`int *ierr`	Returned error flag - should be 0

int idsqna(text)

`char *name`	Object name
`int *ivdsn`	Object handle (if 0, correct handle is returned)
`int *ierr`	Returned error flag - should be 0

int dsshve(name, func, lun)

`char *name`	Object name
`int *ivdsn`	Object handle (if 0, correct handle is returned)
`int *ierr`	Returned error flag - should be 0

int dswidg(name, line, ids, ind)

`char *name`	Object name
`int *ivdsn`	Object handle (if 0, correct handle is returned)
`int *ierr`	Returned error flag - should be 0

int ds_cl_rpn(text)

`char *name`	Object name
`int *ivdsn`	Object handle (if 0, correct handle is returned)
`int *ierr`	Returned error flag - should be 0

int ds_dump(iaddr, imax, int)

`char *name`	Object name
`int *ivdsn`	Object handle (if 0, correct handle is returned)
`int *ierr`	Returned error flag - should be 0

int ds_getarg(i, argi)

`char *name`	Object name
`int *ivdsn`	Object handle (if 0, correct handle is returned)
`int *ierr`	Returned error flag - should be 0

int ds_int_edit(name, nitem, citems,ierr)

`char *name`	Object name
`int *ivdsn`	Object handle (if 0, correct handle is returned)
`int *ierr`	Returned error flag - should be 0

int ids_tapectl(ifd, comm, icnt)

`char *name`	Object name
`int *ivdsn`	Object handle (if 0, correct handle is returned)
`int *ierr`	Returned error flag - should be 0

int dsfndl(namei, nameo, iv)

`char *name`	Object name
`int *ivdsn`	Object handle (if 0, correct handle is returned)
`int *ierr`	Returned error flag - should be 0

1.2.12 Object definition files

int ds_ascii(file_in, echo, ierr)

`char *name`	Object name
`int *ivdsn`	Object handle (if 0, correct handle is returned)
`int *ierr`	Returned error flag - should be 0

1.2.13 Classes

int ds_setcl(name, ivdsn, class, itype, ierr)

`char *name`	Object name
`int *ivdsn`	Object handle (if 0, correct handle is returned)
`int *ierr`	Returned error flag - should be 0

int ds_getcl(name, ivdsn, iclass, itype, ierr)

`char *name`	Object name
`int *ivdsn`	Object handle (if 0, correct handle is returned)
`int *ierr`	Returned error flag - should be 0

int ids_draddr(ivdsn)

`char *name`	Object name
`int *ivdsn`	Object handle (if 0, correct handle is returned)
`int *ierr`	Returned error flag - should be 0

int ids_draddrc(class)

`char *name`	Object name
`int *ivdsn`	Object handle (if 0, correct handle is returned)

`int *ierr`	Returned error flag - should be 0

int ids_ndrivers(ierr)

`char *name`	Object name
`int *ivdsn`	Object handle (if 0, correct handle is returned)
`int *ierr`	Returned error flag - should be 0

int ds_ident_driver(iclass, ident, ierr)

`char *name`	Object name
`int *ivdsn`	Object handle (if 0, correct handle is returned)
`int *ierr`	Returned error flag - should be 0

int ds_obj_mess(message, name, ivdsn, ierr)

`char *name`	Object name
`int *ivdsn`	Object handle (if 0, correct handle is returned)
`int *ierr`	Returned error flag - should be 0

int ids_cls_memb(class, ivdsn, name)

`char *name`	Object name
`int *ivdsn`	Object handle (if 0, correct handle is returned)
`int *ierr`	Returned error flag - should be 0

int ids_drcall(comm, proc, ivdsn, instr)

`char *name`	Object name
`int *ivdsn`	Object handle (if 0, correct handle is returned)
`int *ierr`	Returned error flag - should be 0

int dscprg(text, ivr1, ierr)

`char *name`	Object name
`int *ivdsn`	Object handle (if 0, correct handle is returned)
`int *ierr`	Returned error flag - should be 0

int ds_brdcast(message, iex)

`char *name`	Object name
`int *ivdsn`	Object handle (if 0, correct handle is returned)
`int *ierr`	Returned error flag - should be 0

int ids_nclass(dname)

`char *name` Object name
`int *ivdsn` Object handle (if 0, correct handle is returned)
`int *ierr` Returned error flag - should be 0

int ds_action(comm, name, ivdsn, ierr)

`char *name` Object name
`int *ivdsn` Object handle (if 0, correct handle is returned)
`int *ierr` Returned error flag - should be 0

1.2.14 DSPACK info

int dsvers(dsv, dsd)

`char *name` Object name
`int *ivdsn` Object handle (if 0, correct handle is returned)
`int *ierr` Returned error flag - should be 0

int ids_valid_address(iaddr, ibottom, itop, key)

`char *name` Object name
`int *ivdsn` Object handle (if 0, correct handle is returned)
`int *ierr` Returned error flag - should be 0

1.2.15 User routines

int ds_user_widget(name, value, ipass)

`char *name` Object name
`int *ivdsn` Object handle (if 0, correct handle is returned)
`int *ierr` Returned error flag - should be 0

1.3 Examples