JSOC Reference Manual

@

Generated by Doxygen 1.5.4

Mon Dec 17 19:35:26 2007

Contents

Chapter 1

JSOC Reference

This is the developer and design manual for JSOC. Previous documentation has been integrated into this and it should always be up to date since it is generated directly from the source files using Doxygen.

1.1 General design documents

Blah blah blah

`main JSOC website content` (local)

General information on HMI:

`http://hmi.stanford.edu`

1.2 Hacking on this documentation

There is the beginning of a style guide for documenting under Useful tips for doxygen in C files.

Feel free to start documenting or playing with doxygen configuration. This main page can be found in doc/doxygen_main_page.txt .

This main page is just an introduction to doxygen markup, see the Doxygen manual for the full command set.

- Useful tips for doxygen in C files Tips and hints for using doxygen
- Doxygen reference documentation Links to the Doxygen manual

Chapter 2

JSOC Module Index

2.1 JSOC Modules

Here is a list of all modules:

Chapter 3

JSOC Class Index

3.1 JSOC Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 4

JSOC File Index

4.1 JSOC File List

Here is a list of all documented files with brief descriptions:

Chapter 5

JSOC Page Index

5.1 JSOC Related Pages

Here is a list of all related documentation pages:

Chapter 6

JSOC Module Documentation

6.1 masterlists

6.2 jsoc_main

6.2.1 Detailed Description

Flags:

`-H|-help`: Show jsoc_main usage information.

`-L`: Run jsoc_main driver with logging.

`-V`: Run jsoc_main driver in verbose mode.

`-Q`: Run jsoc_main driver in quiet mode (no terminal output).

Parameters:

DRMS_RETENTION
DRMS_QUERY_MEM
JSOC_DBHOST
JSOC_DBNAME
JSOC_DBUSER
JSOC_DBPASSWD
JSOC_SESSIONNS

Variables

- CmdParams_t **cmdparams**
- DRMS_Env_t * **drms_env**
- ModuleArgs_t * **gModArgs** = module_args

6.3 retrieve_dir

6.3.1 Detailed Description

Retrieve an arbitrary directory of files from SUMS. Typically used to retrieve a directory stored in SUMS with store_dir.

record_set is a database query that allows the user to select a subset of records of a series. In particular, the user can supply values for the *sel* and *note* keywords in *record_set* to search for files saved with store_dir. retrieve_dir retrieves the set of records specified, and for each record uses the value contained within the *dirname_keyword* keyword (defaults to "dirname") to identify the subdirectory within the record's SUMS directory (i.e., it is all of the string after the last '/' in this value) that contains the record's files. It then copies (or links, if the *-l* flag is present) all files from this subdirectory to a identically named subdirectory of the directory specified by *dest*. retrieve_dir will overwrite any files existing in the destination subdirectory whose name matches the name of a file to be copied from a SUMS subdirectory.

If the link flag, *-l*, is specified, then retrieve_dir will create a link to the SUMS subdirectory instead of copying from the SUMS subdirectory to the destination subdirectory. Over time this link may become broken as SUMS may remove a stored file if it is present in SUMS longer than its specified retention time.

Usage:

```
retrieve_dir [-lDRIVER_FLAGS] series=<record_set> to=<dest> [dirkey=<dirname_keyword>]
```

Example: to retrieve a directory of files:

```
retrieve_dir series=su_arta.TestStoreDir to=/home/arta/restoredFilesJanuary/ dirkey=dirname
```

Flags:

- `-l`: Create a symbolic link to the SUMS directory containing the file(s) of interest.

Driver flags: jsoc_main

Parameters:

record_set A series name followed by an optional record set filter (i.e., *name*[*filter*]). Causes selection of a subset of records in the series. Each record retrieved with this *record_set* query will refer to one directory that will be copied to *dest*.

dest The destination directory to which the SUMS subdirectory should be copied/linked. Do not append a final '/' to the path (i.e., /home/arta/dir1 is acceptable, but /home/arta/dir1/ is not)

dirname_keyword Name of the keyword containing the path of the directory originally stored with store_dir.

Variables

- ModuleArgs_t **module_args** [ ]
- char * **module_name** = "retrieve_dir"
- int **verbose** = 0

6.3.2 Variable Documentation

6.3.2.1 ModuleArgs_t module_args[ ]

Initial value:

```
 {
 {ARG_STRING, "series", "", "Series name to retrieve from"},
 {ARG_STRING, "to", "", "Target dir to cp to"},
 {ARG_STRING, "dirkey", "dirname", "Keyword containing the dir name"},
 {ARG_FLAG, "l", "0", "Link instead to the retrieved dir in the SUMS storage"},
 {ARG_END, NULL, NULL}
}
```

Definition at line 99 of file retrieve_dir.c.

6.4 retrieve_file

6.4.1 Detailed Description

Retrieve an arbitrary file from SUMS. Typically used to retrieve a file(s) stored in SUMS with store_file.

record_set is a database query that allows the user to select a subset of records of a series. In particular, the user can supply values for the *sel* and *note* keywords in *record_set* to search for files saved with store_file. For each file in the record set specified by *record_set*, retrieve_file will copy (or link, if the *-l* flag is present) the record segment's file to *dest*. If the series contains more than one segment, *segment_name* must be present. Since the name of a segment's file is the segment name, any file in *dest* whose name matches the segment's name will be overwritten. However, a backup will be saved (the file will be renamed with a $\sim$ suffix) if an overwrite is to occur, unless the *-f* flag is specified. retrieve_file prints to stdout the list of files retrieved, or NOT_FOUND if a file cannot be found. It also prints the contents of the *note* keyword value.

If the link flag, *-l*, is specified, then retrieve_file will create a link to the SUMS file instead of copying the SUMS file to the destination directory. Over time this link may become broken as SUMS may remove a stored file if it is present in SUMS longer than its specified retention time.

Usage:

```
retrieve_file [-fhlvDRIVER_FLAGS] ds=<record_set> out=<dest> [segment=<segment_name>]
```

Example: to retrieve a single file:

```
retrieve_file ds=myseries[][test002]
```

Flags:

`-f`: Force removal of a pre-existing file with same name as retreived file.

`-h`: Print usage message and exit.

`-l`: Create a symbolic link to the SUMS file instead of copying it to *dest*.

`-v`: Run in verbose mode.

Driver flags: jsoc_main

Parameters:

record_set A series name followed by an optional record set filter (i.e., *name*[*filter*]). Causes selection of a subset of records in the series. Each record retrieved with this *record_set* query will refer to one file that will be copied to *dest*.

dest The destination directory to which the SUMS file(s) should be copied/linked. Do not append a final '/' to the path (i.e., /home/arta/dir1 is acceptable, but /home/arta/dir1/ is not)

segment_name Name of the segment whose file should be retrieved.

Defines

- #define **NOTSPECIFIED** "###NOTSPECIFIED###"

Variables

- ModuleArgs_t **module_args** [ ]
- char ∗ **module_name** = "retrieve_file"
- int **verbose** = 0

6.4.2 Variable Documentation

6.4.2.1 ModuleArgs_t module_args[ ]

Initial value:

```
 {
  {ARG_STRING, "segment", NOTSPECIFIED, "", ""},
  {ARG_STRING, "ds", NOTSPECIFIED, "", ""},
  {ARG_STRING, "out", NOTSPECIFIED, "", ""},
  {ARG_FLAG, "f", "0", "", ""},
  {ARG_FLAG, "h", "0", "", ""},
  {ARG_FLAG, "l", "0", "", ""},
  {ARG_FLAG, "v", "0", "", ""},
  {ARG_END, NULL, NULL}
}
```

Definition at line 103 of file retrieve_file.c.

6.5 set_keys

6.5.1 Detailed Description

Modify the keyword values of a DRMS record or create a new record with specified keyword values.

Usage:

```
set_keys [-chmvDRIVER_FLAGS] ds=<record_set> [<keyword1>=<value1>]... [<segment1>=<file1>]...
```

Example: to modify a keyword value:

```
set_keys ds=su_arta.TestStoreFile[file=dsds_data.fits][sel=January] note=fred
```

Example: to create a new record and specify keyword values:

```
set_keys -c ds=su_arta.TestStoreFile file=data.txt sel=February file_seg=/home/arta/febdata.txt
```

Flags:

`-c`: Create a new record

`-h`: Print usage message and exit

`-m`: Modify the keywords of multiple records. The `-m` flag should be used with caution. A typo could damage many records. Do not use `-m` unless you are sure the query will specify ONLY the records you want to modify.

`-v`: Verbose - noisy

The `-c` and `-m` flags cannot be used simultaneously.

Driver flags: jsoc_main

Parameters:

record_set A series name followed by an optional record-set speci- fication (i.e., *series-name*[RecordSet_filter]). If no record-set filter is specified, set_keys requires the *-c* flag, and it creates a new record. All of the prime keywords and values must be specified as *keyword=value* pairs. If a record-set filter IS specified, set_keys requires the *-c* flag to be unset. If *record_set* resolves to more than one record, then the *-m* flag must be set. set_keys will then clone the records specified by the record-set filter. For each keyword specified in a *keyword=value* pair on the command line, set_keys will set the values for all these clones' keywords to *value*.

valueN The new keyword values to be used to create a new or modify an existing record.

fileN If modifying a record(s) and *segmentN* is a generic series segment, then first copy the segment to the cloned record(s)' segment storage, then replace the copied file with *fileN*. Otherwise, a cloned record and its progenitor share the original segment file.

Bug

At present updates of segment files fail. Please use only with the *-c* flag and prime keys when inserting files into generic type data segments.

Variables

- ModuleArgs_t **module_args** [ ]
- char * **module_name** = "set_keys"
- int **verbose** = 0

6.5.2 Variable Documentation

6.5.2.1 ModuleArgs_t module_args[ ]

Initial value:

```
{
  {ARG_STRING, "ds", "Not Specified", "Series name with optional record spec"},
  {ARG_FLAG, "h", "0", "Print usage message and quit"},
  {ARG_FLAG, "c", "0", "Create new record(s) if needed"},
  {ARG_FLAG, "m", "0", "allow multiple records to be updated"},
  {ARG_FLAG, "v", "0", "verbose flag"},
  {ARG_END}
}
```

Definition at line 77 of file set_keys.c.

6.6 show_keys

6.6.1 Detailed Description

Prints keyword information and/or file path for given recordset.

show_keys can list the keyword names and values, and the segment names and file names (full paths) for each record in a record set. It can also list the full path to the record direcory in SUMS, which contains the segment files. Exactly what information gets printed is controlled by command-line flags (see below). The *-k* flag controls the format of the output. If it is set, then the output is in table format, with a header row showing the keyword names. Otherwise, keyword name=value pairs are listed one per line. If the *-a* flag is set, show_keys lists the names of all series keywords, prime keywords, and segments, and exits. Otherwise, it prints keyword and segment information as specified by the other flags and arguments. If the *-p* flag is set and *seglist* is specified, then the full paths for the segment files will be displayed. If the *-p* flag is set, but *seglist* is not specified, then only the full path to the record's storage unit will be displayed.

The number of records for which information will be printed must be specified, either by supplying a *record_set* string that selects a subset of records from a series, or by supplying the *n=nrecords* argument, which indicates the number of records.

Usage:

```
show_keys [-aklpqrDRIVER_FLAGS] ds=<record_set> [n=<nrecords>] [key=<keylist>] [seg=<seglist>]
```

Example: To show the storage-unit paths for a maximum of 10 records:

```
show_keys -p ds=su_arta.TestStoreFile n=10
```

Example: To show information, in non-table format, for all keywords, plus the segment named file_seg, for a maximum of 10 records:

```
show_keys ds=su_arta.TestStoreFile -akr n=10 seg=file_seg
```

Flags:

`-a`: Show all keyword names and values for each record specified by *record_set* or *nrecords*. *-a* takes precedence over *keylist*.

`-k`: List keyword name=value pairs, one per line. Otherwise print all keyword values on a single line and print a header line containing the keyword names (table format).

`-l`: List the names of all series keywords, prime keywords, and segments, and exit. Otherwise, print keyword and segment information as specified by the other flags and arguments.

`-p`: Include in the output the full storage-unit path for each record

`-q`: Quiet - omit the header line listing keyword names if the -k flag is set

`-r`: Include in the output the record number keyword

Driver flags: jsoc_main

Parameters:

record_set A series name followed by an optional record-set specification (i.e., *series-name*[RecordSet_filter]). Causes selection of a subset of records in the series. This argument is required, and if no record-set filter is specified, then *n=nrecords* must be present.

nrecords *nrecords* specifies the maximum number of records for which information is printed. If *nrecords* < 0, show_keys displays information for the last *nrecords* records in the record set. If *nrecords* > 0, show_keys displays information for the first *nrecords* records in the record set. If *record_set* contains a record set filter, then *nrecords* can reduce the total number of records for which information is displayed.

keylist Comma-separated list of keyword names. For each keyword listed, information will be displayed. *keylist* is ignored in the case that the *-a* flag is set.

seglist Comma-separated list of segment names. For each segment listed, the full path to the segment's file is displayed (if the *-p* flag is set) or the file name of the segment's file name is displayed (if the *-p* flag is unset).

Bug

The program will produce superflous and non-meaningful output if called with the *-p* flag and *seglist* is provided on the command line.

See also:

retrieve_file drms_query describe_series

Variables

- ModuleArgs_t **module_args** [ ]
- char * **module_name** = "show_keys"

6.6.2 Variable Documentation

6.6.2.1 ModuleArgs_t module_args[ ]

Initial value:

```
{
  {ARG_STRING, "ds", "Not Specified", "<record_set query>"},
  {ARG_STRING, "key", "Not Specified", "<comma delimited keyword list>"},
  {ARG_STRING, "seg", "Not Specified", "<comma delimited segment list>"},
  {ARG_FLAG, "a", "0", "Show info for all keywords"},
  {ARG_FLAG, "h", "0", "help - print usage info"},
  {ARG_FLAG, "l", "0", "just list series keywords with descriptions"},
  {ARG_INT, "n", "0", "number of records to show, +from first, -from last"},
  {ARG_FLAG, "p", "0", "list the record\'s storage_unit path"},
  {ARG_FLAG, "P", "0", "list the record\'s storage_unit path but no retrieve"},
  {ARG_FLAG, "k", "0", "keyword list one per line"},
  {ARG_FLAG, "q", "0", "quiet - skip header of chosen keywords"},
  {ARG_FLAG, "r", "0", "recnum - show record number as first keyword"},
  {ARG_END}
}
```

Definition at line 119 of file show_keys.c.

6.7 show_series

6.7.1 Detailed Description

List all DRMS dataseries names.

show_series lists the names of DRMS dataseries. If the *-p* flag is set, it displays the prime-keyword names and series description. The information displayed is restricted to a subset of DRMS series by specifying *filter*, a grep-like regular expression.

Flags:

`-h`: Print usage message and exit

`-p`: Print prime-keyword names and the series description

`-v`: Verbose - noisy

Usage:

```
show_series [-hpvDRIVER_FLAGS] [<filter>]
```

Parameters:

filter A pattern using grep-like rules to select a subset of series

Variables

- ModuleArgs_t **module_args** [ ]
- char ∗ **module_name** = "show_series"

6.7.2 Variable Documentation

6.7.2.1 ModuleArgs_t module_args[ ]

Initial value:

```
 {
  {ARG_FLAG, "h", "0", "prints this message and quits."},
  {ARG_FLAG, "p", "0", "enables print prime keys and description."},
  {ARG_FLAG, "v", "0", "verbose"},
  {ARG_END}
}
```

Definition at line 51 of file show_series.c.

6.8 store_dir

6.8.1 Detailed Description

Store arbitrary directory of files to SUMS. Files stored with store_dir can be easily retrieved by the retrieve_dir utility.

If the series *series_name* does not exist, store_dir will first create that series, provided that the user supplies the *-c* flag. If *perm=1*, then the series will be globally accessible, otherwise, only the user calling store_dir will have access. store_dir then creates a record in the series *series_name* and creates a subdirectory, *subdir*, in the record's SUMS directory (for each record there is a single SUMS directory). store_dir copies all files from *prefix/subdir* to *subdir* in the record's SUMS directory. It also stores *prefix/subdir* in the record's *dirname* keyword value so that retrieve_dir can locate the subdirectory within the SUMS directory that contains the files.

sel=*sel_text* and note=*note_text* allow the user to link additional information to the record to facilitate the subsequent search for files in the series. They set the record's *sel* and *note* keywords to have the values *sel_text* and *note_text*, respectively. In particular, *sel* can be used to differentiate between multiple calls to store_dir with the same value for *prefix/subdir*. *series_name* must contain prime keywords *dirname* and *sel*, and it must contain the keyword *note*.

Usage:

```
store_dir [-cvDRIVER_FLAGS] series=<series_name> dirname=<prefix>/<subdir>
                            sel=<sel_text> [note=<note_text>] [perm=0|1]
```

Example: To store a directory of files,

```
store_dir series=su_arta.TestStoreDir dirname=/auto/home2/arta/savedFilesJanuary
          sel=January note="All my January work"
```

Flags:

`-c`: Silently create the series *series_name* if it does not exist.

`-v`: Run in verbose mode.

Driver flags: jsoc_main

Parameters:

series_name Specifies the series, *series_name*, to which the file-referring record will be added. Should a record-set specifier be provided, it will be ignored.

prefix/subdir The local full path that contains the files to be stored in SUMS. The record created will contain a prime keyword named dirname whose value will be *prefix/subdir*.

sel_text Contains a string that will be the keyword value for the *sel* prime keyword of the record created. This extra prime keyword facilitates selection between multiple records containing equivalent values for the dirname (it may be desirable to save files in the same directly repeatedly, or save the same file over time).

note_text An optional string that will be the keyword value for the *note* keyword of the record created.

perm=0|1 Relevant only when the *-c* flag is used and a series is created. A *perm* of 0 will make the created series accessible only by the current user. A *perm* of 1 will make the series globally accessible.

Defines

- #define **NOTSPECIFIED** "***NOTSPECIFIED***"

Variables

- ModuleArgs_t **module_args** [ ]
- char * **module_name** = "store_dir"
- int **verbose** = 0

6.8.2 Variable Documentation

6.8.2.1 ModuleArgs_t module_args[ ]

Initial value:

```
 {
  {ARG_STRING, "series", "", "Series name to store the dir into"},
  {ARG_STRING, "dirname", "", "Dir to store. prime key"},
  {ARG_STRING, "sel", "", "selection name. prime key"},
  {ARG_STRING, "note", "N/A", "comment field"},
  {ARG_INT, "perm", "1", ""},
  {ARG_FLAG, "c", "0", "create new series if needed"},
  {ARG_FLAG, "v", "0", "verbose flag"},
  {ARG_END}
}
```

Definition at line 120 of file store_dir.c.

6.9 store_file

6.9.1 Detailed Description

Store an arbitrary file to SUMS. File stored with store_file can be easily retrieved by the retrieve_file utility.

If the series *series_name* does not exist, store_file will first create that series, provided that the user supplies the *-c* flag. If *perm=1*, then the series will be globally accessible, otherwise, only the user calling store_dir will have access. store_file then creates a record in the series *series_name* and copies the file specified by the full path *file* into the record's SUMS directory (for each record there is a single SUMS directory). store_file stores the filename part of *file* (ie., excludes the path) in the record's *file* keyword value so that retrieve_file can locate the file within the SUMS directory.

sel=*sel_text* and note=*note_text* allow the user to link additional information to the record to facilitate the subsequent search for file in the series. They set the record's *sel* and *note* keywords to have the values *sel_text* and *note_text*, respectively. In particular, *sel* can be used to differentiate between multiple calls to store_file with the same filename part of *file*. *series_name* must contain prime keywords *file* and *sel*, and it must contain the keyword *note*.

Usage:

```
store_file [-chvDRIVER_FLAGS] ds=<series_name> in=<file>
                              [sel=<sel_text>] [note=<note_text>] [perm=0|1]
```

Example: to store a single file:

```
store_file in=0000.fits ds=myseries sel=test002 note= "fits file from test002"
```

Flags:

`-c`: Silently create the series *series_name* if it does not exist.

`-h`: Print usage message and exit

`-v`: Run in verbose mode.

Driver flags: jsoc_main

Parameters:

series_name Specifies the series, *series_name*, to which the file-referring record will be added. Should a record-set specifier be provided, it will be ignored.

file Full path of the file to store. Only the filename (everything after the last '/') will be stored in the *file* keyword.

sel_text Contains a string that will be the value for the *sel* prime keyword of the record created. This extra prime keyword facilitates selection between multiple records containing equivalent values for the *file* keyword (it may be desirable to save files in the same directly repeatedly, or save the same file over time).

note_text An optional string that will be the value for the *note* keyword of the record created.

perm=0|1 Relevant only when the *-c* flag is used and a series is created. A perm of 0 will make the created series accessible only by the current user. A perm of 1 will make the series globally accessible.

Defines

- #define **NOTSPECIFIED** "***NOTSPECIFIED***"

Variables

- ModuleArgs_t **module_args** [ ]
- char * **module_name** = "store_file"
- int **verbose** = 0

6.9.2 Variable Documentation

6.9.2.1 ModuleArgs_t module_args[ ]

Initial value:

```
 {
  {ARG_STRING, "ds", NOTSPECIFIED, "", ""},
  {ARG_STRING, "in", NOTSPECIFIED, "", ""},
  {ARG_STRING, "sel", NOTSPECIFIED, "", ""},
  {ARG_STRING, "note", NOTSPECIFIED, "", ""},
  {ARG_INT, "perm", "1", "", ""},
  {ARG_FLAG, "c", "0", "", ""},
  {ARG_FLAG, "h", "0", "", ""},
  {ARG_FLAG, "v", "0", "", ""},
  {ARG_END}
}
```

Definition at line 124 of file store_file.c.

6.10 DRMS Application Programs

Modules

- masterlists
- drms_server

6.11 drms_server

6.12 DRMS Utilities

Modules

- show_keys
- set_keys
- store_file
- retrieve_file
- store_dir
- retrieve_dir
- create_series
- describe_series
- modify_series
- show_series
- jsoc_main

6.13 create_series

6.14 describe_series

6.15 modify_series

6.16 Core DRMS API functions

- drms_open_records
- drms_create_records
- drms_close_records
- drms_series_exists

Chapter 7

JSOC Class Documentation

7.1 DRMS_Array_struct Struct Reference

`#include <drms_types.h>`

7.1.1 Detailed Description

DRMS array struct.

The DRMS_Array_t data stucture represents an n-dimensional array of scalar data. It is used for internal memory access to data structures read from, or to be written to, record segments. The array data are stored in column-major order at the memory location pointed to by the data element.

The fields israw, bscale, and bzero describe how the data contained in the array data structure relate to the "true" values they are supposed to represent. In the most frequently used case, israw=0, the data stored in memory represent the "true" values of the array, and bzero and bscale contain the shift and scaling (if any) applied to the data when they were read in from external storage. If israw=1, then the data stored in memory represent the unscaled "raw" values of the array, and the true values may be obtained by applying the scaling transformations, if any:

```
f(x) = bzero + bscale * x, if x != MISSING
     = MISSING           , if x == MISSING
```

If the array struct contains data from a DRMS data segment, as returned by the functions drms_segment_readslice or drms_segment_read, then the parent_segment field points to the data segment from which the array data originate.

If the array contains a slice of the parent then the start field contains the starting indices of the slice in the parent array. For example: If an array contains the lower 2x2 elements of a 4x3 data segment then the struct would contain

```
array.naxis = 2
array.axis = [2,2]
array.start = [2,1]
```

Definition at line 506 of file drms_types.h.

Public Attributes

- DRMS_Type_t type

  Datatype of the data elements.

- int naxis

  Number of dimensions.

- int axis [DRMS_MAXRANK]

  Size of each dimension.

- void * data

  Data stored in column major order.

- struct DRMS_Segment_struct * parent_segment

  Parent segment.

- double bzero

  Zero point for parent->child mapping.

- int israw

  Do the values represent true values? Is this read in with type=DRMS_TYPE_RAW? If israw==0 then shift and scaling have been applied to the data and they represent the "true" values. If israw==1 then no shift and scaling have been applied to the data.

- double bscale

  Slope for parent->child.

- int start [DRMS_MAXRANK]

  Start offset of slice in parent.

- int dope [DRMS_MAXRANK]

  Dimension offset multipliers.

- char * strbuf

  String buffer used for packed string arrays.

- long long buflen

  Size of string buffer.

The documentation for this struct was generated from the following file:

- base/drms/libs/api/drms_types.h

7.2 DRMS_Env_struct Struct Reference

`#include <drms_types.h>`

7.2.1 Detailed Description

DRMS environment struct.

Examples:

drms_record_ex1.c.

Definition at line 199 of file drms_types.h.

Public Attributes

- DRMS_Session_t * **session**
- HContainer_t **series_cache**
- HContainer_t **record_cache**
- HContainer_t **storageunit_cache**
- DS_node_t * **templist**
- int **retention**
- int **query_mem**
- int **archive**
- int **server_wait**
- int **verbose**
- int **clientcounter**
- pid_t **tee_pid**
- pthread_mutex_t * **drms_lock**
- pthread_t **sum_thread**
- tqueue_t * **sum_inbox**
- tqueue_t * **sum_outbox**
- long **sum_tag**
- pthread_t **signal_thread**
- sigset_t **signal_mask**
- sigset_t **old_signal_mask**

The documentation for this struct was generated from the following file:

- base/drms/libs/api/drms_types.h

7.3 DRMS_KeyMap_struct Struct Reference

`#include <drms_types.h>`

7.3.1 Detailed Description

DRMS keymap struct.

Examples:

drms_keymap_ex1.c.

Definition at line 805 of file drms_types.h.

Public Attributes

- HContainer_t **int2ext**
- HContainer_t **ext2int**

The documentation for this struct was generated from the following file:

- base/drms/libs/api/drms_types.h

7.4 DRMS_Keyword_struct Struct Reference

`#include <drms_types.h>`

7.4.1 Detailed Description

DRMS keyword struct

Definition at line 406 of file drms_types.h.

Public Attributes

- struct DRMS_Record_struct * **record**
- struct DRMS_KeywordInfo_struct * **info**
- DRMS_Type_Value_t **value**

The documentation for this struct was generated from the following file:

- base/drms/libs/api/drms_types.h

7.5 DRMS_Link_struct Struct Reference

`#include <drms_types.h>`

7.5.1 Detailed Description

DRMS link struct.

Definition at line 441 of file drms_types.h.

Public Attributes

- struct DRMS_Record_struct * **record**
- DRMS_LinkInfo_t * **info**
- long long **recnum**
- int **isset**
- DRMS_Type_Value_t **pidx_value** [DRMS_MAXPRIMIDX]

The documentation for this struct was generated from the following file:

- base/drms/libs/api/drms_types.h

7.6 DRMS_Record_struct Struct Reference

`#include <drms_types.h>`

7.6.1 Detailed Description

DRMS record struct.

Examples:

drms_record_ex5.c.

Definition at line 343 of file drms_types.h.

Public Attributes

- struct DRMS_Env_struct * **env**
- long long **recnum**
- long long **sunum**
- int **init**
- int **readonly**
- DRMS_RecLifetime_t **lifetime**
- struct DRMS_StorageUnit_struct * **su**
- int **slotnum**
- long long **sessionid**
- char * **sessionns**
- DRMS_SeriesInfo_t * **seriesinfo**
- HContainer_t **keywords**
- HContainer_t **links**
- HContainer_t **segments**

The documentation for this struct was generated from the following file:

- base/drms/libs/api/drms_types.h

7.7 DRMS_Segment_struct Struct Reference

`#include <drms_types.h>`

7.7.1 Detailed Description

DRMS segment struct.

A DRMS data segment corresponds to a named file, typically containing an n-dimensional scalar array. (It can also be a "generic" segment, which is just an unstructured file as far as DRMS is concerned.) One or more segments constitute the external data part(s) of the DRMS record pointed to by the record field. The info field points to a structure containing attributes common to all records in a series, while the segment structure itself contains the fields axis and blocksize that can vary from record to record if scope=DRMS_VARDIM.

The protocol field determines the external storage format used for storing segment data. Only protocols DRMS_BINARY, DRMS_BINZIP, DRMS_FITS, DRMS_FITZ, DRMS_GENERIC, and DRMS_TAS are fully supported in the base DRMS system (NetDRMS). Protocol DRMS_DSDS is a special protocol for dealing with the format of the Stanford SOI-MDI Data Sorage and Distribution System (DSDS) and requires support outside the DRMS library. Protocol DRMS_LOCAL likewise supports the DSDS file-format and requires a non-NetDRMS library. It differs from DRMS_DSDS in that it does not depend on the presence of DSDS - it merely allows the user to operate on files external to DSDS (eg., files that may reside on a LOCAL hard disk) that happen to have the file format that DSDS uses. DRMS_GENERIC and DRMS_MSI are also reserved for unsupported data formats. In particular, the DRMS_GENERIC protocol is used to refer to any unstructured data format or data formats of unknown structure.

Data storage for DRMS_FITS is in minimal simple FITS files, without extensions and with only the compliance- and structure-defining keywords (SIMPLE, BITPIX, NAXIS, NAXISn, and END, and optionally BLANK, BSCALE and BZERO) in the headers. All other ancillary data are to be found in the DRMS record. For the DRMS_FITZ protocol, the representation is similar except that the entire FITS file is compressed with Rice compression. (Note that because the memory representation for the data supported through the API functions drms_segment_read is the DRMS_Array_t struct, which has a maximum rank of 16, FITS hypercubes of dimension > 16 are not supported.)

For the DRMS_BINARY protocol, the data are written in a binary format, in which the first 8 bytes are the characters "DRMS RAW", the next `8(n+1)` are little-endian integer representations of the data type, rank, and dimensions of the *n* axes, and the remainder the binary data in little-endian format. For the DRMS_BINZIP protocol the represntation is the same, except that the file is gzip compressed. The DRMS_TAS protocol (for "Tiled Array Storage") is described elsewhere, if at all. It is designed for use with data segments that are small compared with the size of the full data records, in order to minimize file access without keeping all of the segment data in the relational database, by concatenating multiple segments in the external format. The segment blocksize member is for use with the DRMS_TAS protocol.

Segment data types refer to the scalar data type for the segment, and should be mostly self-explanatory. DRMS_TYPE_TIME is a special case of double-precision floating point values representing elapsed time from a fixed epoch. Arithmetic is the same as for DRMS_TYPE_DOUBLE, only the format for string representations differs from that for normal floating-point data; see sprint_time. Data of type DRMS_TYPE_STRING are null-terminated byte sequences of any length. Data type DRMS_TYPE_STRING is not supported by the protocols DRMS_BINARY, DRMS_BINZIP, DRMS_FITS, nor DRMS_FITZ. Whether it is properly supported by the DRMS_TAS protocol is doubtful. The data type DRMS_TYPE_RAW is used to describe data that are not to be converted on read from the type of their external representation, which must then be established for type-specific operations. It should be used for DRMS_Array_t structures only, not for DRMS segments.

The scope of a segment can take on three values. The normal scope is expected to be DRMS_VARIABLE,

for which the particular segment for every record has exactly the same structure (rank and dimensions), only the actual data values vary from one record to another. (Note that different segments of a record, however, need not have the same structure as one another.) If the scope is DRMS_VARDIM, then the dimensions and even rank of the particular segment may vary from one record to another, although other features of the segment, in particular the data type, must still be the same. Scope DRMS_CONSTANT is used to describe a data segment that is constant for all records. It can be used for example to describe a location index array, or a constant calibration array that applies to all records in the series, so that it can be made available to any record without having to store multiple instances externally.

Definition at line 756 of file drms_types.h.

Public Attributes

- struct DRMS_Record_struct * record

  The record this segment belongs to.

- DRMS_SegmentInfo_t * info

  Contains attributes common to all records in a series.

- char filename [DRMS_MAXSEGFILENAME]

  Storage file name.

- int axis [DRMS_MAXRANK]

  Size of each dimension.

- int blocksize [DRMS_MAXRANK]

  Block sizes for TAS storage.

The documentation for this struct was generated from the following file:

- base/drms/libs/api/drms_types.h

7.8 DRMS_SegmentDimInfo_struct Struct Reference

`#include <drms_types.h>`

7.8.1 Detailed Description

DRMS segment dimension info struct.

Definition at line 629 of file drms_types.h.

Public Attributes

- int naxis
  Number of dimensions (rank).
- int axis [DRMS_MAXRANK]
  Length of each dimension.

The documentation for this struct was generated from the following file:

- base/drms/libs/api/drms_types.h

7.9 DRMS_SegmentInfo_struct Struct Reference

`#include <drms_types.h>`

7.9.1 Detailed Description

DRMS segment info struct.

Definition at line 640 of file drms_types.h.

Public Attributes

- char name [DRMS_MAXNAMELEN]
  Segment name.
- int segnum
  Segment number in record.
- char description [DRMS_MAXCOMMENTLEN]
  Description string.
- int islink
  Non-0 if segment inherited.
- char linkname [DRMS_MAXNAMELEN]
  Link to inherit from.
- char target_seg [DRMS_MAXNAMELEN]
  Segment to inherit.
- DRMS_Type_t type
  Datatype of data elements.
- int naxis
  Number of dimensions (rank).
- char unit [DRMS_MAXUNITLEN]
  Physical unit.
- DRMS_Protocol_t protocol
  Storage protocol.
- DRMS_Segment_Scope_t scope
  Const, Varies, or DimsVary.
- long long cseg_recnum
  Record number where constant segment is stored.

The documentation for this struct was generated from the following file:

- base/drms/libs/api/drms_types.h

7.10 DRMS_Session_struct Struct Reference

`#include <drms_types.h>`

7.10.1 Detailed Description

DRMS Session struct.

Definition at line 152 of file drms_types.h.

Public Attributes

- int **db_direct**
- DB_Handle_t * **db_handle**
- DB_Handle_t * **stat_conn**
- long long **sessionid**
- char * **sessionns**
- long long **sunum**
- char * **sudir**
- int **clientid**
- char **hostname** [DRMS_MAXHOSTNAME]
- unsigned short **port**
- int **sockfd**

The documentation for this struct was generated from the following file:

- base/drms/libs/api/drms_types.h

7.11 DRMS_Type_Value Union Reference

`#include <drms_types.h>`

7.11.1 Detailed Description

DRMS type value.

Definition at line 89 of file drms_types.h.

Public Attributes

- char **char_val**
- short **short_val**
- int **int_val**
- long long **longlong_val**
- float **float_val**
- double **double_val**
- double **time_val**
- char * **string_val**

The documentation for this union was generated from the following file:

- base/drms/libs/api/drms_types.h

7.12 HContainer_struct Struct Reference

`#include <hcontainer.h>`

7.12.1 Detailed Description

HContainer struct.

Examples:

drms_series_ex3.c.

Definition at line 24 of file hcontainer.h.

Public Attributes

- int **num_total**
- int **datasize**
- int **keysize**
- Hash_Table_t **hash**
- void(* **deep_free**)(const void *value)
- void(* **deep_copy**)(const void *dst, const void *src)
- HCBuf_t * **buf**

The documentation for this struct was generated from the following file:

- base/libs/dstruct/hcontainer.h

Chapter 8

JSOC File Documentation

8.1 base/drms/libs/api/drms_array.h File Reference

8.1.1 Detailed Description

Functions to access DRMS array data structures.

See also:

drms_record.h drms_keyword.h drms_link.h drms_segment.h drms_types.h

Definition in file drms_array.h.

```
#include "drms_statuscodes.h"
#include "drms_priv.h"
```

Defines

- #define **INLINE** static inline
- #define **DRMS_ARRAY2STRING_LEN** 30
- #define **DRMS_ARRAY_GETVAL**(VAL, X, Y)
- #define **DRMS_ARRAY_SETVAL**(VAL, X, Y)

Functions

- INLINE void **drms_array_setv** (DRMS_Array_t *arr,...)

Information and Diagnostics

- INLINE int drms_array_offset (DRMS_Array_t *arr, int *indexarr)
- void drms_array_print (DRMS_Array_t *arr, const char *colsep, const char *rowsep)
- INLINE long long drms_array_count (DRMS_Array_t *arr)
- INLINE long long drms_array_size (DRMS_Array_t *arr)
- INLINE int drms_array_naxis (DRMS_Array_t *arr)
- INLINE int drms_array_nth_axis (DRMS_Array_t *arr, int n)

Filling

- INLINE int drms_array_setchar_ext (DRMS_Array_t *arr, long long index, char value)
- INLINE int **drms_array_setchar** (DRMS_Array_t *arr, int *indexarr, char value)
- INLINE int **drms_array_setshort_ext** (DRMS_Array_t *arr, long long index, short value)
- INLINE int **drms_array_setshort** (DRMS_Array_t *arr, int *indexarr, short value)
- INLINE int **drms_array_setint_ext** (DRMS_Array_t *arr, long long index, int value)
- INLINE int **drms_array_setint** (DRMS_Array_t *arr, int *indexarr, int value)
- INLINE int **drms_array_setlonglong_ext** (DRMS_Array_t *arr, long long index, long long value)
- INLINE int **drms_array_setlonglong** (DRMS_Array_t *arr, int *indexarr, long long value)
- INLINE int **drms_array_setfloat_ext** (DRMS_Array_t *arr, long long index, float value)
- INLINE int **drms_array_setfloat** (DRMS_Array_t *arr, int *indexarr, float value)
- INLINE int **drms_array_setdouble_ext** (DRMS_Array_t *arr, long long index, double value)
- INLINE int **drms_array_setdouble** (DRMS_Array_t *arr, int *indexarr, double value)
- INLINE int drms_array_settime_ext (DRMS_Array_t *arr, long long index, double value)
- INLINE int drms_array_settime (DRMS_Array_t *arr, int *indexarr, double value)
- INLINE int drms_array_setstring_ext (DRMS_Array_t *arr, long long index, char *value)
- INLINE int drms_array_setstring (DRMS_Array_t *arr, int *indexarr, char *value)
- INLINE int drms_array_setext (DRMS_Array_t *arr, long long index, DRMS_Value_t *src)
- INLINE int drms_array_set (DRMS_Array_t *arr, int *indexarr, DRMS_Value_t *src)
- void drms_array2missing (DRMS_Array_t *arr)

Creation and Destruction

- DRMS_Array_t * drms_array_create (DRMS_Type_t type, int naxis, int *axis, void *data, int *status)
- void drms_free_array (DRMS_Array_t *src)

Slicing and Permutation

- DRMS_Array_t * drms_array_slice (int *start, int *end, DRMS_Array_t *src)
- DRMS_Array_t * drms_array_permute (DRMS_Array_t *src, int *perm, int *status)

Scaling and Type Conversion

- int drms_array2char (int n, DRMS_Type_t src_type, double bzero, double bscale, void *src, char *dst)
- int **drms_array2short** (int n, DRMS_Type_t src_type, double bzero, double bscale, void *src, short *dst)
- int **drms_array2int** (int n, DRMS_Type_t src_type, double bzero, double bscale, void *src, int *dst)
- int **drms_array2longlong** (int n, DRMS_Type_t src_type, double bzero, double bscale, void *src, long long *dst)
- int **drms_array2float** (int n, DRMS_Type_t src_type, double bzero, double bscale, void *src, float *dst)
- int **drms_array2double** (int n, DRMS_Type_t src_type, double bzero, double bscale, void *src, double *dst)
- int drms_array2time (int n, DRMS_Type_t src_type, double bzero, double bscale, void *src, double *dst)
- int drms_array2string (int n, DRMS_Type_t src_type, double bzero, double bscale, void *src, char **dst)
- DRMS_Array_t * drms_array_convert (DRMS_Type_t dsttype, double bzero, double bscale, DRMS_Array_t *src)
- void drms_array_convert_inplace (DRMS_Type_t newtype, double bzero, double bscale, DRMS_Array_t *src)
- int drms_array_rawconvert (int n, DRMS_Type_t dsttype, double bzero, double bscale, void *dst, DRMS_Type_t srctype, void *src)

8.1.2 Function Documentation

8.1.2.1 int drms_array2char (int *n*, DRMS_Type_t *src_type*, double *bzero*, double *bscale*, void * *src*, char * *dst*)

Convert the first *n* values of *src*, interpreted as being of type *src_type*, to values of the destination data type (the data type of *dst*) and place the values in *dst*. The values are first scaled by *bzero* and *bscale*. If the scaled values are not within the range representable by the destination datatype, they are replaced by the destination-datatype-specific missing value; otherwise, the scaled values are placed in *dst*. (This is almost certainly a bug!) If *src_type* is DRMS_TYPE_STRING, the data strings are interpreted as character representations of numbers with *strtod*, and then scaled. If the resulting values are within the representable range for the destination datatype, they are placed in *dst*, otherwise the appropriate missing value is used. (This appears to be the only case for which these functions behave as expected.)

Parameters:

n Number of values to convert.

src_type Source datatype (see DRMS_Type_t).

bzero Offset by which raw data values are to be shifted to produce actual data values.

bscale Scaling factor by which raw data values are to be multiplied to produce actual data values.

src Source data, which is of type *srctype*.

dst Contiguous array of the destination data type into which the converted data are stored upon success.

Returns:

DRMS status (see drms_statuscodes.h). 0 if successful, non-0 otherwise.

Definition at line 338 of file drms_array.c.

References DRMS_MISSING_CHAR, DRMS_MISSING_INT, DRMS_MISSING_LONGLONG, DRMS_MISSING_SHORT, DRMS_MISSING_TIME, DRMS_TYPE_CHAR, DRMS_TYPE_DOUBLE, DRMS_TYPE_FLOAT, DRMS_TYPE_INT, DRMS_TYPE_LONGLONG, DRMS_TYPE_SHORT, DRMS_TYPE_STRING, and DRMS_TYPE_TIME.

8.1.2.2 void drms_array2missing (DRMS_Array_t * *arr*)

Sets all the values in the array *arr->data* to the DRMS entity representing missing (fill) data for the data type *arr->type*, for example DRMS_MISSING_SHORT or DRMS_MISSING_DOUBLE. If the data type is DRMS_TYPE_STRING, the values are set to single-byte null terminators. Note that the value defined for DRMS_MISSING_TIME is in fact a valid double-precision number, and that this value is defined differently from the definitions used for representation of invalid strings in sscan_time.

Parameters:

arr The DRMS array struct whose values are set to missing.

Definition at line 191 of file drms_array.c.

References DRMS_Array_struct::data, drms_array_count(), DRMS_MISSING_CHAR, DRMS_MISSING_DOUBLE, DRMS_MISSING_FLOAT, DRMS_MISSING_INT, DRMS_MISSING_LONGLONG, DRMS_MISSING_SHORT, DRMS_MISSING_TIME, DRMS_TYPE_CHAR, DRMS_TYPE_DOUBLE, DRMS_TYPE_FLOAT, DRMS_TYPE_INT, DRMS_TYPE_LONGLONG, DRMS_TYPE_SHORT, DRMS_TYPE_STRING, DRMS_TYPE_TIME, and DRMS_Array_struct::type.

8.1.2.3 int drms_array2string (int *n*, DRMS_Type_t *src_type*, double *bzero*, double *bscale*, void * *src*, char ** *dst*)

Converts the first *n* values of *src*, interpreted as being of type *src_type*, to strings which are placed in the array *dst* using a *sprintf* function with a 24.17lg format, if the type is numeric. If the type is DRMS_TYPE_TIME, the *sprint_time* function is used, with a TAI representation accurate to the nearest second. If the type is DRMS_TYPE_STRING, the strings are simply copied. For numeric types (including DRMS_TYPE_TIME), the data are first scaled by *bzero* and *bscale* before being printed to strings. Data with values represent- ing missing data are represented by single-character null terminators; however, no check is performed on whether the scaling would result in valid data for the type.

Parameters:

n Number of values to convert.

src_type Source datatype (see DRMS_Type_t).

bzero Offset by which raw data values are to be shifted to produce actual data values.

bscale Scaling factor by which raw data values are to be multiplied to produce actual data values.

src Source data, which is of type *srctype*.

dst Contiguous array of string pointers into which the converted data are stored upon success.

Returns:

DRMS status (see drms_statuscodes.h). 0 if successful, non-0 otherwise.

Definition at line 2590 of file drms_array.c.

References DRMS_MISSING_CHAR, DRMS_MISSING_INT, DRMS_MISSING_LONGLONG, DRMS_MISSING_SHORT, DRMS_MISSING_TIME, DRMS_TYPE_CHAR, DRMS_TYPE_DOUBLE, DRMS_TYPE_FLOAT, DRMS_TYPE_INT, DRMS_TYPE_LONGLONG, DRMS_TYPE_SHORT, DRMS_TYPE_STRING, and DRMS_TYPE_TIME.

8.1.2.4 int drms_array2time (int *n*, DRMS_Type_t *src_type*, double *bzero*, double *bscale*, void * *src*, double * *dst*)

Converts the first *n* values of *src*, interpreted as being of type *src_type*, to *doubles*, scaling them by *bzero* and *bscale*. If the resulting scaled numeric values are outside the representable range for the type, the value is replaced by DRMS_MISSING_TIME. If the data are of type DRMS_TYPE_STRING, then they are scanned by sscan_time, then scaled, and then checked for validity as *doubles*.

Parameters:

n Number of values to convert.

src_type Source datatype (see DRMS_Type_t).

bzero Offset by which raw data values are to be shifted to produce actual data values.

bscale Scaling factor by which raw data values are to be multiplied to produce actual data values.

src Source data, which is of type *srctype*.

dst Contiguous array of the time datatype values into which the converted data are stored upon success.

Returns:

DRMS status (see drms_statuscodes.h). 0 if successful, non-0 otherwise.

Definition at line 2296 of file drms_array.c.

References DRMS_MISSING_CHAR, DRMS_MISSING_INT, DRMS_MISSING_LONGLONG, DRMS_MISSING_SHORT, DRMS_MISSING_TIME, DRMS_TYPE_CHAR, DRMS_TYPE_-DOUBLE, DRMS_TYPE_FLOAT, DRMS_TYPE_INT, DRMS_TYPE_LONGLONG, DRMS_TYPE_-SHORT, DRMS_TYPE_STRING, and DRMS_TYPE_TIME.

8.1.2.5 DRMS_Array_t* drms_array_convert (DRMS_Type_t *dsttype*, double *bzero*, double *bscale*, DRMS_Array_t * *src*)

Converts the data in *src->data* to type *dsttype*, scaling by the values *bzero* and *bscale*, by calling the function drms_array_rawconvert. It returns a newly created array struct with the converted values but without copying or setting the elements other than those set by drms_array_create. In particular, the elements *bzero* and *bscale* are not set. drms_array_convert_inplace performs the same type conversion, but instead of returning a new array simply replaces the *src->data* element.

Parameters:

dst Contiguous array of type *dsttype* into which the converted data are stored upon success.

bzero Offset by which raw data values are to be shifted to produce actual data values.

bscale Scaling factor by which raw data values are to be multiplied to produce actual data values.

src The source array containing that data to be converted.

Definition at line 83 of file drms_array.c.

References DRMS_Array_struct::axis, DRMS_Array_struct::data, drms_array_count(), DRMS_Array_-struct::naxis, and DRMS_Array_struct::type.

8.1.2.6 void drms_array_convert_inplace (DRMS_Type_t *newtype*, double *bzero*, double *bscale*, DRMS_Array_t * *src*)

Performs the same type conversion as drms_array_convert, but instead of returning a new array simply replaces the *src->data* element.

Parameters:

dst Contiguous array of type *dsttype* into which the converted data are stored upon success.

bzero Offset by which raw data values are to be shifted to produce actual data values.

bscale Scaling factor by which raw data values are to be multiplied to produce actual data values.

src The source array containing that data to be converted.

Definition at line 66 of file drms_array.c.

References DRMS_Array_struct::data, and DRMS_Array_struct::type.

8.1.2.7 INLINE long long drms_array_count (DRMS_Array_t * *arr*)

Returns the total number of data points in the array *arr->data*, i.e. the product of *arr->axis*[i].

Parameters:

arr The DRMS array struct whose data points are to be counted.

Returns:

The number of data points in *arr*.

Definition at line 171 of file drms_array.h.

References DRMS_Array_struct::axis, and DRMS_Array_struct::naxis.

Referenced by drms_array2missing(), drms_array_convert(), drms_array_print(), drms_array_size(), and drms_segment_autoscale().

8.1.2.8 DRMS_Array_t* drms_array_create (DRMS_Type_t *type*, int *naxis*, int * *axis*, void * *data*, int * *status*)

Creates a DRMS_Array_t struct with the specified values for the elements *type*, *naxis*, and *axis*. The array *axis* must be of length *naxis* (at least) unless it is NULL. If *data* is a nun-NULL pointer, the data pointer of the created array is set to it; otherwise, a data space of appropriate size is malloc'd (but left with unknown contents). *type* cannot be DRMS_TYPE_RAW. No checks are made for legitimate (positive) values of *axis*[n]. None of the other elements of the struct are filled except *dope*.

Parameters:

type The data type of the DRMS array struct to create.

naxis The number of axes of the DRMS array struct to create.

axis The lengths of each axis of the DRMS array struct to create.

data A linear array of data values, of type *type*, which are "stolen" by the DRMS array struct being created.

DRMS status (see drms_statuscodes.h). 0 if successful, non-0 otherwise.

Returns:

The created DRMS array struct.

Definition at line 7 of file drms_array.c.

References DRMS_Array_struct::axis, DRMS_Array_struct::data, DRMS_Array_struct::dope, drms_array_size(), DRMS_MAXRANK, DRMS_TYPE_RAW, DRMS_Array_struct::naxis, and DRMS_Array_struct::type.

8.1.2.9 INLINE int drms_array_naxis (DRMS_Array_t * *arr*)

Returns the rank of the array, *arr->naxis*.

Parameters:

arr The DRMS array struct whose number of axes is being counted.

Returns:

The number of axes in *arr*.

Definition at line 204 of file drms_array.h.

References DRMS_Array_struct::naxis.

8.1.2.10 INLINE int drms_array_nth_axis (DRMS_Array_t * *arr*, int *n*)

Returns the dimension of axis *n* of the array, *arr->axis*[n].

Parameters:

arr The DRMS array struct whose dimension length is being returned.

n The axis number of *arr* whose length is being returned.

Returns:

The length of axis *n*.

Definition at line 218 of file drms_array.h.

References DRMS_Array_struct::axis.

8.1.2.11 INLINE int drms_array_offset (DRMS_Array_t * *arr*, int * *indexarr*)

Returns the offset, in bytes, from the start of the array *arr->data* of the datum at the coordinate value specified by the index array *indexarr*, which must be of dimension *arr->naxis* (at least).

Parameters:

arr The DRMS array struct for which the offset is being calculated.

indexarr The index into *arr* of the datum for which the offset is being calculated.

Returns:

The bytes offset of the datum at *indexarr*.

Definition at line 138 of file drms_array.h.

References DRMS_Array_struct::dope, and DRMS_Array_struct::naxis.

Referenced by drms_array_set(), drms_array_setstring(), and drms_array_settime().

8.1.2.12 DRMS_Array_t* drms_array_permute (DRMS_Array_t * *src*, int * *perm*, int * *status*)

Rearranges the array elements in *arr->data* such that the dimensions are ordered according to the permutation given in the vector *perm* (dimension *arr->naxis*), using the function *ndim_perm*. This is a generalization of the matrix transpose operator to n dimensions. The permuted data are returned in a newly created array struct, created wih drms_array_create.

Parameters:

src The DRMS array struct whose data will be transposed.

perm Vector specifying the order in which data should be transposed.

status DRMS status (see drms_statuscodes.h). 0 if successful, non-0 otherwise.

Returns:

The created DRMS array struct.

Definition at line 127 of file drms_array.c.

References DRMS_Array_struct::axis, DRMS_Array_struct::data, DRMS_Array_struct::naxis, and DRMS_Array_struct::type.

8.1.2.13 void drms_array_print (DRMS_Array_t * *arr*, const char * *colsep*, const char * *rowsep*)

Prints the values of the data in *arr->data* in tabular form, with colums separated by the string *colsep* and rows separated by the string *rowsep*. One-dimensional arrays are printed as a single row, two-dimansional arrays in column-major *i.e.* storage order. Arrays of higher dimension are printed as successive tables, each labeled by a header line giving the array index or indices for the corresponding dimension(s) above the 2nd.

Parameters:

arr The DRMS array struct whose data values are being printed.

colsep A string that will be printed between columns of output.

rowsep A string that will be printed between rows of output.

Definition at line 263 of file drms_array.c.

References DRMS_Array_struct::axis, DRMS_Array_struct::data, drms_array_count(), DRMS_MAXRANK, DRMS_Array_struct::naxis, and DRMS_Array_struct::type.

8.1.2.14 int drms_array_rawconvert (int *n*, DRMS_Type_t *dsttype*, double *bzero*, double *bscale*, void * *dst*, DRMS_Type_t *srctype*, void * *src*)

Converts the first *n* values of *src->data*, interpreted as being of type *srctype*, to type *dsttype*, with scaling by *bzero* and *bscale* as applicable for the datatype, by calling the appropriate function *drms_array2**. The resulting data are placed in *dst*.

Parameters:

n Number of values to convert.

dsttype Destination datatype (see DRMS_Type_t).

bzero Offset by which raw data values are to be shifted to produce actual data values.

bscale Scaling factor by which raw data values are to be multiplied to produce actual data values.

dst Contiguous array of type *dsttype* into which the converted data are stored upon success.

srctype Source datatype (see DRMS_Type_t).

src Source data, which is of type *srctype*.

Returns:

DRMS status (see drms_statuscodes.h). 0 if successful, non-0 otherwise.

Definition at line 151 of file drms_array.c.

References DRMS_TYPE_CHAR, DRMS_TYPE_DOUBLE, DRMS_TYPE_FLOAT, DRMS_TYPE_INT, DRMS_TYPE_LONGLONG, DRMS_TYPE_SHORT, DRMS_TYPE_STRING, and DRMS_TYPE_TIME.

8.1.2.15 INLINE int drms_array_set (DRMS_Array_t * *arr*, int * *indexarr*, DRMS_Value_t * *src*)

Sets the value of the *arr->data* array element indexed by the vector *indexarr*, of length *arr->naxis* (at least) to the value *src*, converted to the type *src->type* by *drms2** as necessary.

Parameters:

arr The DRMS array struct whose array element is set.

indexarr The array of index values that specify an array element whose value is set.

src The to which the designed array element is set.

Returns:

DRMS status (see drms_statuscodes.h). 0 if successful, non-0 otherwise.

Definition at line 628 of file drms_array.h.

References drms_array_offset(), and drms_array_setext().

8.1.2.16 INLINE int drms_array_setchar_ext (DRMS_Array_t * *arr*, long long *index*, char *value*)

Set an array element in manner similar to drms_array_set and drms_array_setext, except do not perform a type conversion; the array data type *arr->type* must match the type of *value*.

Parameters:

arr The DRMS array struct whose array element is set.

index The index of the array element whose value is set.

value The to value which the designed array element is set.

Returns:

DRMS status (see drms_statuscodes.h). 0 if successful, non-0 otherwise.

Definition at line 242 of file drms_array.h.

References DRMS_Array_struct::data, DRMS_ERROR_INVALIDDATA, DRMS_TYPE_CHAR, and DRMS_Array_struct::type.

8.1.2.17 INLINE int drms_array_setext (DRMS_Array_t * *arr*, long long *index*, DRMS_Value_t * *src*)

Sets the value of the array element indexed by *index* from the start of the array *arr->data* (in units of the size of the array data type) to the value *src*, converted to the type *src->type* by *drms2** as necessary.

Parameters:

arr The DRMS array struct whose array element is set.

index The index of the array element whose value is set.

src The to which the designed array element is set.

Returns:

DRMS status (see drms_statuscodes.h). 0 if successful, non-0 otherwise.

Definition at line 501 of file drms_array.h.

References DRMS_Array_struct::data, DRMS_TYPE_CHAR, DRMS_TYPE_DOUBLE, DRMS_TYPE_FLOAT, DRMS_TYPE_INT, DRMS_TYPE_LONGLONG, DRMS_TYPE_SHORT, DRMS_TYPE_STRING, DRMS_TYPE_TIME, and DRMS_Array_struct::type.

Referenced by drms_array_set().

8.1.2.18 INLINE int drms_array_setstring (DRMS_Array_t * *arr*, int * *indexarr*, char * *value*)

Set an array element in manner similar to drms_array_set and drms_array_setext, except do not perform a type conversion; the array data type *arr->type* must match the type of *value*. This function places the start address of the string in the array, it does not do a string copy.

Parameters:

arr The DRMS array struct whose array element is set.

index The index of the array element whose value is set.

value The string to which the designed array element is set.

Returns:

DRMS status (see drms_statuscodes.h). 0 if successful, non-0 otherwise.

Definition at line 475 of file drms_array.h.

References DRMS_Array_struct::data, drms_array_offset(), DRMS_ERROR_INVALIDDATA, DRMS_TYPE_STRING, and DRMS_Array_struct::type.

8.1.2.19 INLINE int drms_array_setstring_ext (DRMS_Array_t * *arr*, long long *index*, char * *value*)

Set an array element in manner similar to drms_array_set and drms_array_setext, except do not perform a type conversion; the array data type *arr->type* must match the type of *value*. This function places the start address of the string in the array, it does not do a string copy.

Parameters:

arr The DRMS array struct whose array element is set.

index The index of the array element whose value is set.

value The string to which the designed array element is set.

Returns:

DRMS status (see drms_statuscodes.h). 0 if successful, non-0 otherwise.

Definition at line 450 of file drms_array.h.

References DRMS_Array_struct::data, DRMS_ERROR_INVALIDDATA, DRMS_TYPE_STRING, and DRMS_Array_struct::type.

8.1.2.20 INLINE int drms_array_settime (DRMS_Array_t * *arr*, int * *indexarr*, double *value*)

Set an array element in manner similar to drms_array_set and drms_array_setext, except do not perform a type conversion; the array data type *arr->type* must be DRMS_TYPE_TIME).

Parameters:

arr The DRMS array struct whose array element is set.

index The index of the array element whose value is set.

value The to time which the designed array element is set.

Returns:

DRMS status (see drms_statuscodes.h). 0 if successful, non-0 otherwise.

Definition at line 424 of file drms_array.h.

References DRMS_Array_struct::data, drms_array_offset(), DRMS_ERROR_INVALIDDATA, DRMS_TYPE_TIME, and DRMS_Array_struct::type.

8.1.2.21 INLINE int drms_array_settime_ext (DRMS_Array_t * *arr*, long long *index*, double *value*)

Set an array element in manner similar to drms_array_set and drms_array_setext, except do not perform a type conversion; the array data type *arr->type* must be DRMS_TYPE_TIME).

Parameters:

arr The DRMS array struct whose array element is set.

index The index of the array element whose value is set.

value The time to which the designed array element is set.

Returns:

DRMS status (see drms_statuscodes.h). 0 if successful, non-0 otherwise.

Definition at line 402 of file drms_array.h.

References DRMS_Array_struct::data, DRMS_ERROR_INVALIDDATA, DRMS_TYPE_TIME, and DRMS_Array_struct::type.

8.1.2.22 INLINE long long drms_array_size (DRMS_Array_t * *arr*)

Returns the total size in bytes of the data array *arr->data*, i.e. the product of drms_array_count and drms_sizeof(*arr->type*).

Parameters:

arr The DRMS array struct whose size is being calculated.

Returns:

The byte size of *arr*.

Definition at line 190 of file drms_array.h.

References drms_array_count(), and DRMS_Array_struct::type.

Referenced by drms_array_create(), and drms_segment_read().

8.1.2.23 DRMS_Array_t* drms_array_slice (int * *start*, int * *end*, DRMS_Array_t * *src*)

Returns a newly created array struct (created with drms_array_create, q.v.) in which the data element corresponds to a subset hypercube of dimension *src->naxis* with only the index values for dimension *n* in the closed range [*start*[n], *end*[n]]. The extraction is performed by the function ndim_pack.

Parameters:

start The index value of the dimension (orthogonal to the slice) that marks the start of the slice.

end The index value of the dimension (orthogonal to the slice) that marks the end of the slice.

src The DRMS array struct from which a slice is to be extracted.

Returns:

The created DRMS array struct.

Definition at line 99 of file drms_array.c.

References DRMS_Array_struct::axis, DRMS_Array_struct::data, DRMS_MAXRANK, DRMS_Array_-struct::naxis, and DRMS_Array_struct::type.

8.1.2.24 void drms_free_array (DRMS_Array_t * *src*)

Frees the array struct *src* as well as its member *src->data* as necessary.

Parameters:

src The DRMS array struct to free.

Definition at line 55 of file drms_array.c.

References DRMS_Array_struct::data.

8.2 base/drms/libs/api/drms_keymap.h File Reference

8.2.1 Detailed Description

Functions to create and free DRMS_KeyMap_t structures.

See also:

drms_keyword.h

Definition in file drms_keymap.h.

Functions

- DRMS_KeyMap_t * drms_keymap_create (void)
- void drms_keymap_destroy (DRMS_KeyMap_t **km)
- int drms_keymap_parsetable (DRMS_KeyMap_t *keymap, const char *text)
- int drms_keymap_parsefile (DRMS_KeyMap_t *keymap, FILE *fPtr)

8.2.2 Function Documentation

8.2.2.1 DRMS_KeyMap_t* drms_keymap_create (void)

Allocate an empty DRMS_KeyMap_t and return a pointer to it if memory was successfully allocated. It is the caller's responsiblity to free the returned DRMS_KeyMap_t by calling drms_keymap_destroy.

Returns:

Pointer to a newly allocated DRMS_KeyMap_t structure.

Examples:

drms_keymap_ex1.c.

Definition at line 149 of file drms_keymap.c.

References DRMS_MAXNAMELEN, DRMS_KeyMap_struct::ext2int, and DRMS_KeyMap_-struct::int2ext.

8.2.2.2 void drms_keymap_destroy (DRMS_KeyMap_t ** *km*)

Free all allocated memory associated with a DRMS_KeyMap_t structure. The DRMS_KeyMap_t pointer contained in *km* is set to NULL.

Parameters:

km Pointer to a pointer to the DRMS_KeyMap_t structure being freed.

Examples:

drms_keymap_ex1.c.

Definition at line 167 of file drms_keymap.c.

8.2.2.3 int drms_keymap_parsefile (DRMS_KeyMap_t * *keymap*, FILE * *fPtr*)

Parse a file containing FITS-keyword-name-to-DRMS-keyword-name mappings. *fPtr* must contain a valid file pointer. If *keymap* is not NULL, the resulting set of mappings is used to initialize *keymap*. This function calls drms_keymap_parsetable to parse the content of the file to which *fPtr* refers.

Parameters:

keymap Pointer to an existing DRMS_KeyMap_t structure. Upon success, this structure will be initialzed with keyword pairs specified in the content of the file to which *fPtr* refers.

fPtr File pointer referring to a file containing keyword-pair specifications.

Returns:

1 if successful, 0 otherwise

Definition at line 263 of file drms_keymap.c.

8.2.2.4 int drms_keymap_parsetable (DRMS_KeyMap_t * *keymap*, const char * *text*)

Parse a buffer containing FITS-keyword-name-to-DRMS-keyword-name mappings. The buffer, *text*, can contain zero or more mappings, each of the form *fitsname*(*whitespace* | ',')*drmsname*. Each pair of mappings must be separated by a newline character ('\n'). Comments or emtpy strings may appear between newline characters as well. If *keymap* is not NULL, the resulting set of mappings is used to initialize *keymap*.

Parameters:

keymap Pointer to an existing DRMS_KeyMap_t structure. Upon success, this structure will be initialzed with keyword pairs specified in *text*.

text Buffer containing keyword mappings.

Returns:

1 if successful, 0 otherwise

Examples:

drms_keymap_ex1.c.

Definition at line 185 of file drms_keymap.c.

References DRMS_MAXNAMELEN, DRMS_KeyMap_struct::ext2int, and DRMS_KeyMap_struct::int2ext.

8.3 base/drms/libs/api/drms_keyword.h File Reference

8.3.1 Detailed Description

Functions to access DRMS keyword values, and to convert DRMS keywords to FITS keywords and vice versa.

Definition in file drms_keyword.h.

```
#include "drms_types.h"
```

Functions

- void **drms_free_template_keyword_struct** (DRMS_Keyword_t *key)
- void **drms_free_keyword_struct** (DRMS_Keyword_t *key)
- void **drms_copy_keyword_struct** (DRMS_Keyword_t *dst, DRMS_Keyword_t *src)
- HContainer_t * **drms_create_keyword_prototypes** (DRMS_Record_t *target, DRMS_Record_t *source, int *status)
- void **drms_keyword_print** (DRMS_Keyword_t *key)
- void **drms_keyword_printval** (DRMS_Keyword_t *key)
- int **drms_template_keywords** (DRMS_Record_t *template)
- DRMS_Keyword_t * **drms_keyword_lookup** (DRMS_Record_t *rec, const char *key, int followlink)
- DRMS_Type_t **drms_keyword_type** (DRMS_Keyword_t *key)
- HContainer_t * **drms_keyword_createinfocon** (DRMS_Env_t *drmsEnv, const char *seriesName, int *status)
- void **drms_keyword_destroyinfocon** (HContainer_t **info)
- int **drms_keyword_keysmatch** (DRMS_Keyword_t *k1, DRMS_Keyword_t *k2)
- char **drms_getkey_char** (DRMS_Record_t *rec, const char *key, int *status)
- short **drms_getkey_short** (DRMS_Record_t *rec, const char *key, int *status)
- int **drms_getkey_int** (DRMS_Record_t *rec, const char *key, int *status)
- long long **drms_getkey_longlong** (DRMS_Record_t *rec, const char *key, int *status)
- float **drms_getkey_float** (DRMS_Record_t *rec, const char *key, int *status)
- double **drms_getkey_double** (DRMS_Record_t *rec, const char *key, int *status)
- char * **drms_getkey_string** (DRMS_Record_t *rec, const char *key, int *status)
- DRMS_Type_Value_t **drms_getkey** (DRMS_Record_t *rec, const char *key, DRMS_Type_t *type, int *status)
- int **drms_setkey_char** (DRMS_Record_t *rec, const char *key, char value)
- int **drms_setkey_short** (DRMS_Record_t *rec, const char *key, short value)
- int **drms_setkey_int** (DRMS_Record_t *rec, const char *key, int value)
- int **drms_setkey_longlong** (DRMS_Record_t *rec, const char *key, long long value)
- int **drms_setkey_float** (DRMS_Record_t *rec, const char *key, float value)
- int **drms_setkey_double** (DRMS_Record_t *rec, const char *key, double value)
- int **drms_setkey_string** (DRMS_Record_t *rec, const char *key, const char *value)
- int **drms_setkey** (DRMS_Record_t *rec, const char *key, DRMS_Type_t type, DRMS_Type_Value_t *value)
- int **drms_setkey_p** (DRMS_Record_t *rec, const char *key, DRMS_Value_t *value)
- DRMS_Value_t **drms_getkey_p** (DRMS_Record_t *rec, const char *key, int *status)
- int **drms_copykey** (DRMS_Record_t *target, DRMS_Record_t *source, const char *key)
- int **drms_keyword_getintname** (const char *keyname, char *nameOut, int size)

- int **drms_keyword_getintname_ext** (const char *keyname, DRMS_KeyMapClass_t *classid, DRMS_KeyMap_t *map, char *nameOut, int size)
- int **drms_keyword_getextname** (DRMS_Keyword_t *key, char *nameOut, int size)
- int **drms_keyword_getextname_ext** (DRMS_Keyword_t *key, DRMS_KeyMapClass_t *class, DRMS_KeyMap_t *map, char *nameOut, int size)

8.4 base/drms/libs/api/drms_protocol.h File Reference

8.4.1 Detailed Description

Definition in file drms_protocol.h.

```
#include "drms_types.h"
```

Classes

- struct **DRMS_GenericSpecific_struct**
- struct **DRMS_BinarySpecific_struct**
- struct **DRMS_BinzipSpecific_struct**
- struct **DRMS_FitsSpecific_struct**
- struct **DRMS_FitzSpecific_struct**
- struct **DRMS_MSISpecific_struct**
- struct **DRMS_TASSpecific_struct**

Typedefs

- typedef struct DRMS_GenericSpecific_struct **DRMS_GenericSpecific_t**
- typedef struct DRMS_BinarySpecific_struct **DRMS_BinarySpecific_t**
- typedef struct DRMS_BinzipSpecific_struct **DRMS_BinzipSpecific_t**
- typedef struct DRMS_FitsSpecific_struct **DRMS_FitsSpecific_t**
- typedef struct DRMS_FitzSpecific_struct **DRMS_FitzSpecific_t**
- typedef struct DRMS_MSISpecific_struct **DRMS_MSISpecific_t**
- typedef struct DRMS_TASSpecific_struct **DRMS_TASSpecific_t**

Enumerations

- enum DRMS_Protocol_t {
  DRMS_GENERIC, DRMS_BINARY, DRMS_BINZIP, DRMS_FITZ,
  DRMS_FITS, DRMS_MSI, DRMS_TAS, DRMS_DSDS,
  DRMS_LOCAL }
  DRMS segment protocols.

Functions

- DRMS_Protocol_t **drms_str2prot** (const char *str)
- const char * **drms_prot2str** (DRMS_Protocol_t prot)

8.4.2 Enumeration Type Documentation

8.4.2.1 enum DRMS_Protocol_t

DRMS segment protocols.

Enumerator:

DRMS_GENERIC Arbitrary file format which can vary across records.

DRMS_BINARY Binary file format.

DRMS_BINZIP Binary file format which is gzip compressed.

DRMS_FITZ Simple FITS file format which is gzip compressed.

DRMS_FITS Simple FITS file format.

DRMS_MSI Unsupported.

DRMS_TAS "Tiled Array Storage" file format

DRMS_DSDS DSDS file format stored in DSDS (read only).

DRMS_LOCAL DSDS file format stored locally (read only).

Definition at line 14 of file drms_protocol.h.

8.5 base/drms/libs/api/drms_record.h File Reference

8.5.1 Detailed Description

Functions that retrieve, close, populate, copy, allocate, and free DRMS_Record_t structures.

See also:

drms_keymap.h drms_keyword.h drms_segment.h drms_series.h drms_env.h

Definition in file drms_record.h.

```
#include "drms_types.h"
#include "db.h"
```

Defines

- #define **kLocalSegName** "local_data"
- #define **kLocalPrimekey** "primekey"
- #define **kLocalPrimekeyType** DRMS_TYPE_LONGLONG

Enumerations

- enum **DRMS_CloneAction_t** { **DRMS_COPY_SEGMENTS**, **DRMS_SHARE_SEGMENTS** }
- enum **DRMS_CloseAction_t** { **DRMS_FREE_RECORD**, **DRMS_INSERT_RECORD** }

Functions

- DRMS_RecordSet_t * drms_open_records (DRMS_Env_t *env, char *recordsetname, int *status)
- DRMS_RecordSet_t * **drms_open_localrecords** (DRMS_Env_t *env, const char *dsRecSet, int *status)
- DRMS_RecordSet_t * **drms_open_dsdsrecords** (DRMS_Env_t *env, const char *dsRecSet, int *status)
- DRMS_RecordSet_t * drms_clone_records (DRMS_RecordSet_t *recset, DRMS_RecLifetime_t lifetime, DRMS_CloneAction_t mode, int *status)
- DRMS_RecordSet_t * drms_create_records (DRMS_Env_t *env, int n, char *seriesname, DRMS_RecLifetime_t lifetime, int *status)
- DRMS_RecordSet_t * **drms_create_records_fromtemplate** (DRMS_Env_t *env, int n, DRMS_Record_t *template, DRMS_RecLifetime_t lifetime, int *status)
- DRMS_RecordSet_t * **drms_create_recprotos** (DRMS_RecordSet_t *recset, int *status)
- void **drms_destroy_recprotos** (DRMS_RecordSet_t **protos)
- DRMS_Record_t * **drms_create_recproto** (DRMS_Record_t *recSource, int *status)
- void **drms_destroy_recproto** (DRMS_Record_t **proto)
- int drms_close_records (DRMS_RecordSet_t *rs, int action)
- DRMS_Record_t * **drms_clone_record** (DRMS_Record_t *record, DRMS_RecLifetime_t lifetime, DRMS_CloneAction_t mode, int *status)
- DRMS_Record_t * **drms_create_record** (DRMS_Env_t *env, char *seriesname, DRMS_RecLifetime_t lifetime, int *status)
- int **drms_close_record** (DRMS_Record_t *rec, int action)
- void **drms_print_record** (DRMS_Record_t *rec)

- long long **drms_record_size** (DRMS_Record_t *rec)
- int **drms_record_numkeywords** (DRMS_Record_t *rec)
- int **drms_record_numlinks** (DRMS_Record_t *rec)
- int **drms_record_numsegments** (DRMS_Record_t *rec)
- int **drms_record_num_nonlink_segments** (DRMS_Record_t *rec)
- void **drms_record_directory** (DRMS_Record_t *rec, char *dirname, int retrieve)
- int **drms_recproto_setseriesinfo** (DRMS_Record_t *rec, int *unitSize, int *bArchive, int *nDaysRetention, int *tapeGroup, const char *description)
- DRMS_RecordQueryType_t **drms_record_getquerytype** (const char *query)
- long long **drms_record_memsize** (DRMS_Record_t *rec)
- char * **drms_record_jsoc_version** (DRMS_Env_t *env, DRMS_Record_t *rec)

8.5.2 Function Documentation

8.5.2.1 DRMS_RecordSet_t* drms_clone_records (DRMS_RecordSet_t * *recset*, DRMS_RecLifetime_t *lifetime*, DRMS_CloneAction_t *mode*, int * *status*)

Create a new set of records using values from an original set of records to populate the new records. Within the current DRMS session, this function creates a record-set structure (DRMS_RecordSet_t) that contains "copies" of the record structures contained in *recset*. For each record in *recset*, a new DRMS_Record_t structure is created and assigned a unique record number from the DRMS database. The values of the keywords, links, and segments of the original record in recset are used to populate the newly created record. If *mode* == DRMS_SHARE_SEGMENTS, the newly created segments will share the segment files with the original record, i.e. the new record will have the same storage unit number (DSINDEX) as the original record. However, keyword and link data will be replicated. If *mode* == DRMS_COPY_SEGMENTS, the original record's segment files will be copied to a new storage unit slot, and the copied files will be associated with the new record.

The newly created records are placed in the record cache (DRMS_Env_t->record_cache) and are made writeable and assigned a lifetime of *lifetime* (please see drms_reclifetime for details on lifetime).

Upon successful completion, the function returns a DRMS_RecordSet_t pointer, and sets **status* to 0. If an error occurs, the function returns NULL and sets **status* to an appropriate error code defined in drms_statuscodes.h. Typical errors are as follows. If *recset* does not have any legitimate records, then **status* is set to DRMS_ERROR_BADRECORDCOUNT. If there was an error receiving one or more proper record numbers from the database server, then **status* is set to DRMS_ERROR_BADSEQUENCE. If an error occurs while creating a SUMS slot directory, then *status is set to DRMS_ERROR_MKDIRFAILED.

The caller owns the allocated memory associated with the returned record set and must release it by calling drms_close_records.

Parameters:

recset The original set of records that get cloned.

lifetime Either DRMS_PERMANENT (at the end of the DRMS session, the cloned records should be saved to the database) or DRMS_TRANSIENT (at the end of the DRMS session, the cloned records should be discarded).

mode Either DRMS_COPY_SEGMENTS (copy original data to the newly cloned records) or DRMS_SHARE_SEGMENTS(point to the original data).

status Pointer to DRMS status (see drms_statuscodes.h) returned by reference. 0 if successful, non-0 otherwise.

Returns:

The set of cloned records.

Examples:

drms_record_ex3.c, and drms_record_ex4.c.

Definition at line 1689 of file drms_record.c.

References DRMS_Segment_struct::axis, DRMS_Segment_struct::blocksize, DRMS_TAS, DRMS_-TYPE_RAW, DRMS_Segment_struct::info, DRMS_SegmentInfo_struct::name, DRMS_SegmentInfo_-struct::naxis, DRMS_SegmentInfo_struct::protocol, DRMS_Record_struct::recnum, DRMS_Record_-struct::segments, DRMS_Record_struct::seriesinfo, DRMS_Env_struct::session, DRMS_Session_-struct::sessionid, DRMS_Session_struct::sessionns, DRMS_Record_struct::slotnum, DRMS_Record_-struct::su, DRMS_Record_struct::sunum, and DRMS_SegmentInfo_struct::type.

8.5.2.2 int drms_close_records (DRMS_RecordSet_t * *rs*, int *action*)

Close a set of records and free allocated memory, optionally inserting new records into the database. If *action* == DRMS_FREE_RECORD, then if a record being closed is the only reference to a SUMS storage unit slot, that slot is freed and marked for removal during SUMS garbage collection. During SUMS garbage collection, all data-segment files stored within this storage unit will be deleted. In this scenario, segment files written to SUMS during the current DRMS session are not preserved. If action == DRMS_INSERT_-RECORD, then this function saves the keyword, link, and segment information in the appropriate database tables, and ensures that segment files written to SUMS during the current DRMS session are preserved. In order to succeed, no record contained in rs can be marked read-only.

Regardless of *action*, this function calls drms_free_records to deallo- cate the *rs* DRMS_RecordSet_t structure, the array of pointers to DRMS_Record_t structures contained within *rs*, and the actual DRMS_-Record_t structures that these pointers reference. All these structures and pointers must have been previously allocated in heap memory (functions like drms_open_records ensure this is the case). The records are also removed from the record cache (*env->record_cache*).

Upon successful completion, the function returns 0. If an error occurs, the function returns an appropriate error code defined in drms_statuscodes.h. Typical errors are as follows. If action is nei- ther DRMS_-FREE_RECORD nor DRMS_INSERT_RECORD, then DRMS_ERROR_INVALI- DACTION is returned. If *action* == DRMS_INSERT_RECORD and at least one record in *rs* is marked read-only, then DRMS_-ERROR_COMMITREADONLY is returned.

Parameters:

rs The set of records to close

action Specifes whether records being closed are to be inserted into the database (DRMS_INSERT_-RECORD) or not (DRMS_FREE_RECORD).

Returns:

DRMS status (see drms_statuscodes.h). 0 if successful, non-0 otherwise.

Examples:

drms_record_ex2.c, drms_record_ex3.c, and drms_record_ex4.c.

Definition at line 1884 of file drms_record.c.

References DRMS_Record_struct::env, DRMS_Record_struct::readonly, DRMS_Record_-struct::seriesinfo, DRMS_Record_struct::slotnum, and DRMS_Record_struct::su.

8.5.2.3 DRMS_RecordSet_t* drms_create_records (DRMS_Env_t * *env*, int *n*, char * *seriesname*, DRMS_RecLifetime_t *lifetime*, int * *status*)

Create a new set of *n* records for series *seriesname*. Within the current DRMS session, this function creates a record-set structure (DRMS_RecordSet_t) that contains *n* newly created record structures (DRMS_Record_t). Each created record is assigned a unique record number from the DRMS database. The values of the keywords, links, and segments from the series' template record (stored in the series cache (DRMS_Env_t->record_cache) are used to populate the corresponding values of each of the *n* created records.

The newly created records are placed in the record cache (DRMS_Env_t->record_cache) and are made writeable and assigned a lifetime of *lifetime* (please see drms_reclifetime for details on lifetime).

Upon successful completion, the function returns a DRMS_RecordSet_t pointer, and sets **status* to 0. If an error occurs, the function returns NULL and sets **status* to an appropriate error code defined in drms_statuscodes.h. Typical errors are as follows. If there was an error receiving one or more proper record numbers from the database server, then **status* is set to DRMS_ERROR_BADSEQUENCE. If an error occurs while creating a SUMS slot directory, then *status is set to DRMS_ERROR_MKDIRFAILED.

The caller owns the allocated memory associated with the returned record set and must release it by calling drms_close_records.

Parameters:

env Contains information about the DRMS session in which the records should be created.

n Number of records to create.

seriesname Name of the series into which records should be inserted.

lifetime Either DRMS_PERMANENT (at the end of the DRMS session, the created records should be saved to the database) or DRMS_TRANSIENT (at the end of the DRMS session, the created records should be discarded).

status Pointer to DRMS status (see drms_statuscodes.h) returned by reference. 0 if successful, non-0 otherwise.

Returns:

The set of created records.

Definition at line 1378 of file drms_record.c.

8.5.2.4 DRMS_RecordSet_t* drms_open_records (DRMS_Env_t * *env*, char * *recordsetname*, int * *status*)

Retrieve a set of records specified by a recordset query. Within the current DRMS session (whose information is stored in *env*), this function submits a database query specified in *recordsetname* and creates a record-set structure (DRMS_RecordSet_t) to contain the results of the query. If, at the time this function is called, a requested record structure (DRMS_Record_t) exists in the record cache (*env->record_cache*), then a pointer to that record structure is inserted into the results record set. Otherwise, a new record structure is created, populated from the database, inserted into the record cache, and inserted into the results record set. The newly created record is marked read-only and assigned a permanent lifetime (DRMS_PER- MANENT).

Upon successful completion, the function returns a DRMS_RecordSet_t pointer, and sets **status* to DRMS_SUCCESS. If an error occurs, the function returns NULL and sets **status* to an appropriate error code defined in drms_statuscodes.h. Typical errors are as follows. If a problem occurs during communication with the database server, **status* is set to DRMS_ERROR_QUERYFAILED. If the number of records to be returned exceeds the allowable number, then **status* is set to DRMS_QUERY_TRUNCATED.

The caller owns the allocated memory associated with the returned record set and must release it by calling drms_close_records.

Parameters:

env DRMS session information.

recordsetname A string that specifies a database query. It includes a series name and clauses to extract a subset of records from that series. Please see http://jsoc.stanford.edu/jsocwiki/DrmsNames for more information about database queries.

status Pointer to DRMS status (see drms_statuscodes.h) returned by reference. 0 if successful, non-0 otherwise.

Returns:

The set of records retrieved by the query.

Examples:

drms_record_ex2.c, drms_record_ex3.c, and drms_record_ex4.c.

Definition at line 1161 of file drms_record.c.

References DRMS_ERROR_INVALIDDATA, DRMS_ERROR_OUTOFMEMORY, and HContainer_-struct::num_total.

8.6 base/drms/libs/api/drms_record_priv.h File Reference

8.6.1 Detailed Description

Private functions that retrieve, close, populate, copy, allocate, and free DRMS_Record_t structures.

See also:

drms_keymap.h drms_keyword.h drms_segment.h drms_series.h drms_env.h

Definition in file drms_record_priv.h.

```
#include "drms_types.h"
```

Functions

- int drms_closeall_records (DRMS_Env_t *env, int action)
- FILE * drms_record_fopen (DRMS_Record_t *rec, char *filename, const char *mode)
- void drms_free_record (DRMS_Record_t *rec)
- void drms_free_records (DRMS_RecordSet_t *rs)
- long long drms_record_size (DRMS_Record_t *rec)
- DRMS_Record_t * drms_retrieve_record (DRMS_Env_t *env, const char *seriesname, long long recnum, int *status)
- DRMS_RecordSet_t * drms_retrieve_records (DRMS_Env_t *env, const char *seriesname, char *where, int filter, int mixed, int *status)
- int drms_insert_records (DRMS_RecordSet_t *recset)
- DRMS_Record_t * drms_template_record (DRMS_Env_t *env, const char *seriesname, int *status)
- int drms_populate_record (DRMS_Record_t *record, long long recnum)
- int drms_populate_records (DRMS_RecordSet_t *rs, DB_Binary_Result_t *qres)
- DRMS_Record_t * drms_alloc_record (DRMS_Env_t *env, const char *series, long long recnum, int *status)
- DRMS_Record_t * drms_alloc_record2 (DRMS_Record_t *template, long long recnum, int *status)
- char * drms_field_list (DRMS_Record_t *rec, int *num_args)
- void drms_copy_record_struct (DRMS_Record_t *dst, DRMS_Record_t *src)
- void drms_free_record_struct (DRMS_Record_t *rec)
- void drms_free_template_record_struct (DRMS_Record_t *rec)

8.6.2 Function Documentation

8.6.2.1 DRMS_Record_t* drms_alloc_record (DRMS_Env_t * *env*, const char * *series*, long long *recnum*, int * *status*)

xxx

Definition at line 2369 of file drms_record.c.

8.6.2.2 DRMS_Record_t* drms_alloc_record2 (DRMS_Record_t * *template*, long long *recnum*, int * *status*)

xxx

Definition at line 2334 of file drms_record.c.

References DRMS_MAXHASHKEYLEN, DRMS_Record_struct::recnum, DRMS_Env_struct::record_cache, and DRMS_Record_struct::su.

8.6.2.3 int drms_closeall_records (DRMS_Env_t * *env*, int *action*)

Close all records in the record cache, optionally inserting modified, writeable records into the database. This function iterates through all records in the record cache (""DRMS_Env_trecord_cache). If *action* == DRMS_INSERT_RECORD, then for each such record that is not marked read-only, this function calls drms_close_records, passing the record as an argument. The result is preservation of the record's data in the database. If *action* == DRMS_FREE_RECORD, then the record's data is discarded. Please see drms_close_records for more details. For each record in the record cache, this function calls drms_free_record to deallocate the DRMS_Record_t structure associated with the record, and it removes the record from the record cache.

Parameters:

env DRMS session information.

action If DRMS_INSERT_RECORD, save all modifiable records in the database, if DRMS_FREE_RECORD, do not save any record.

Returns:

DRMS status (see drms_statuscodes.h). 0 if successful, non-0 otherwise.

Examples:

drms_record_ex1.c.

Definition at line 2041 of file drms_record.c.

References DRMS_Record_struct::readonly, and DRMS_Env_struct::record_cache.

8.6.2.4 void drms_copy_record_struct (DRMS_Record_t * *dst*, DRMS_Record_t * *src*)

xxx

Definition at line 2578 of file drms_record.c.

References DRMS_Record_struct::keywords, DRMS_Record_struct::links, DRMS_Segment_struct::record, DRMS_Keyword_struct::record, DRMS_Link_struct::record, and DRMS_Record_struct::segments.

8.6.2.5 char* drms_field_list (DRMS_Record_t * *rec*, int * *num_args*)

xxx

Definition at line 2779 of file drms_record.c.

References DRMS_VARDIM, DRMS_Segment_struct::info, DRMS_Keyword_struct::info, DRMS_Link_struct::info, DRMS_Record_struct::keywords, DRMS_Record_struct::links, DRMS_SegmentInfo_struct::naxis, DRMS_SegmentInfo_struct::scope, and DRMS_Record_struct::segments.

8.6.2.6 void drms_free_record (DRMS_Record_t * *rec*)

Remove record *rec* from the DRMS-session-specific record cache (DRMS_Env_t::record_cache), freeing all allocated memory associated with the record. drms_free_record_struct is the function that deallocates this memory. Removal decrements the ref count on the record's storage-unit directory. When the ref count reaches 0, the entire storage unit is freed with a call to drms_freeunit.

Parameters:

rec Pointer to the DRMS record to be removed and freed.

Definition at line 2065 of file drms_record.c.

References DRMS_MAXHASHKEYLEN, DRMS_Record_struct::env, DRMS_Record_struct::recnum, DRMS_Env_struct::record_cache, and DRMS_Record_struct::seriesinfo.

8.6.2.7 void drms_free_record_struct (DRMS_Record_t * *rec*)

xxx

Definition at line 2552 of file drms_record.c.

References DRMS_Session_struct::db_direct, DRMS_Record_struct::env, DRMS_Record_struct::init, DRMS_Record_struct::keywords, DRMS_Record_struct::links, DRMS_Record_struct::segments, DRMS_Env_struct::session, DRMS_Record_struct::sessionns, and DRMS_Record_struct::su.

8.6.2.8 void drms_free_records (DRMS_RecordSet_t * *rs*)

Remove all records contained in *rs* from the DRMS-session-specific record cache (DRMS_Env_t::record_-cache), freeing all allocated memory associated with each record. Make *rs->n* calls to drms_free_record. Also frees *rs->records*, an array of DRMS_Record_t * and *rs*.

Parameters:

rs Pointer to the DRMS recordset containing records to be removed.

Definition at line 1945 of file drms_record.c.

8.6.2.9 void drms_free_template_record_struct (DRMS_Record_t * *rec*)

xxx

Definition at line 2532 of file drms_record.c.

References DRMS_Record_struct::init, DRMS_Record_struct::keywords, DRMS_Record_struct::links, DRMS_Record_struct::segments, and DRMS_Record_struct::seriesinfo.

8.6.2.10 int drms_insert_records (DRMS_RecordSet_t * *recset*)

xxx

Definition at line 2911 of file drms_record.c.

References DRMS_Segment_struct::axis, DRMS_MAXNAMELEN, DRMS_TYPE_INT, DRMS_-TYPE_LONGLONG, DRMS_TYPE_STRING, DRMS_VARDIM, DRMS_Record_struct::env, DRMS_-Segment_struct::filename, DRMS_Segment_struct::info, DRMS_Keyword_struct::info, DRMS_Link_-struct::info, DRMS_Link_struct::isset, DRMS_Record_struct::keywords, DRMS_Record_struct::links, DRMS_SegmentInfo_struct::naxis, DRMS_Link_struct::pidx_value, DRMS_Link_struct::recnum, DRMS_Record_struct::recnum, DRMS_SegmentInfo_struct::scope, DRMS_Record_struct::segments, DRMS_Record_struct::seriesinfo, DRMS_Env_struct::session, DRMS_Record_struct::sessionid, DRMS_Record_struct::sessionns, DRMS_Record_struct::slotnum, DRMS_Type_Value::string_val, DRMS_Record_struct::sunum, and DRMS_Keyword_struct::value.

8.6.2.11 int drms_populate_record (DRMS_Record_t * *record*, long long *recnum*)

xxx

Definition at line 2611 of file drms_record.c.

References DRMS_MAXNAMELEN, DRMS_Record_struct::env, DRMS_Record_struct::seriesinfo, and DRMS_Env_struct::session.

8.6.2.12 int drms_populate_records (DRMS_RecordSet_t * *rs*, DB_Binary_Result_t * *qres*)

xxx

Definition at line 2668 of file drms_record.c.

References DRMS_Segment_struct::axis, DRMS_MAXSEGFILENAME, DRMS_VARDIM, DRMS_-Segment_struct::filename, DRMS_Segment_struct::info, DRMS_Keyword_struct::info, DRMS_Link_-struct::info, DRMS_Link_struct::isset, DRMS_Record_struct::keywords, DRMS_Record_struct::links, DRMS_SegmentInfo_struct::naxis, DRMS_Link_struct::pidx_value, DRMS_Link_struct::recnum, DRMS_Record_struct::recnum, DRMS_Segment_struct::record, DRMS_Keyword_struct::record, DRMS_Link_struct::record, DRMS_SegmentInfo_struct::scope, DRMS_Record_struct::segments, DRMS_SegmentInfo_struct::segnum, DRMS_Record_struct::sessionid, DRMS_Record_struct::sessionns, DRMS_Record_struct::slotnum, DRMS_Record_struct::sunum, and DRMS_Keyword_struct::value.

8.6.2.13 FILE* drms_record_fopen (DRMS_Record_t * *rec*, char * *filename*, const char * *mode*)

Open a file named *filename*, with mode *mode*, in the storage-unit directory that belongs to record *rec*. If **mode* == 'w' or **mode* == 'a' and *rec* has not been assigned a storage-unit directory, one is allocated. If **mode* == 'r' *rec* must have an assigned storage-unit directory. *rec* must contain a least one DRMS data segment.

Parameters:

rec (DRMS_Record_t *) Pointer to a DRMS record that contains at least one data segment.

filename (char *) Name of the file that exists or will be created in the storage-unit directory belonging to *rec*.

mode (const char *) Pointer to a *char* that specifies the mode to pass to *fopen*. Supported values include 'a', 'r', and 'w'.

Returns:

(FILE *) Upon success, returns a *FILE* * to the file opened. Otherwise, returns NULL.

Definition at line 3327 of file drms_record.c.

References DRMS_Record_struct::env, DRMS_Record_struct::lifetime, DRMS_Record_struct::recnum, DRMS_Record_struct::seriesinfo, DRMS_Record_struct::slotnum, DRMS_Record_struct::su, and DRMS_Record_struct::sunum.

8.6.2.14 long long drms_record_size (DRMS_Record_t * *rec*)

Return the actual size in bytes of the data contained in *rec*.

Parameters:

rec Pointer to the DRMS record whose data size is being requested.

Returns:

The number of data bytes contained in *rec*.

Definition at line 3160 of file drms_record.c.

References DRMS_Record_struct::segments.

8.6.2.15 DRMS_Record_t* drms_retrieve_record (DRMS_Env_t * *env*, const char * *seriesname*, long long *recnum*, int * *status*)

xxx

Definition at line 2085 of file drms_record.c.

References DRMS_MAXHASHKEYLEN, DRMS_Record_struct::readonly, and DRMS_Env_struct::record_cache.

8.6.2.16 DRMS_RecordSet_t* drms_retrieve_records (DRMS_Env_t * *env*, const char * *seriesname*, char * *where*, int *filter*, int *mixed*, int * *status*)

xxx

Definition at line 2151 of file drms_record.c.

References DRMS_MAXHASHKEYLEN, DRMS_Env_struct::query_mem, DRMS_Env_struct::record_cache, and DRMS_Env_struct::session.

8.6.2.17 DRMS_Record_t* drms_template_record (DRMS_Env_t * *env*, const char * *seriesname*, int * *status*)

Within the current DRMS session (whose information is stored in *env*), this function returns a template record for the series *seriesname*. If such a template record exists in the series template-record cache (`env->series_cache`), that one is returned. Otherwise a new template record is created, cached, and returned to the caller. The template-record structure is allocated and assigned default values derived from entries in the global database tables: *drms_series*, *drms_segment*, *drms_link*, and *drms_keyword*. drms_template_record calls drms_template_segments, drms_template_links, and drms_template_keywords to populate segment, link, and keyword values.

The caller does not own the allocated memory associated with the returned record and should not release it (i.e., the caller should not call drms_close_record).

Typical uses of this function include obtaining a list of keywords or a particular keyword, primary or otherwise, obtaining a list of segments or a particular segment, obtaining a list of links or a particular link, and obtaining series information.

This really should be private, and not available to modules. If a module needs series information, it should call drms_series.h functions.

Parameters:

env DRMS session information.

seriesname The name of the series for which the template record is desired.

status DRMS status (see drms_statuscodes.h). 0 if successful, non-0 otherwise.

Returns:

Upon successful completion, the function returns a DRMS_Record_t pointer, and sets ∗*status* to DRMS_SUCCESS. If an error occurs, the function returns NULL and sets ∗*status* to an appropriate error code defined in drms_statuscodes.h. Typical errors are as follows. If *seriesname* does not exist in the series cache, then ∗*status* is set to DRMS_ERROR_UNKNOWNSERIES. If a problem occurs during communication with the database server, ∗*status* is set to DRMS_ERROR_QUERYFAILED. If a database table is malformed or corrupt, ∗*status* is set to DRMS_ERROR_BADQUERYRESULT.

Examples:

drms_record_ex5.c.

Definition at line 2393 of file drms_record.c.

References DRMS_ERROR_UNKNOWNSERIES, DRMS_MAXCOMMENTLEN, DRMS_-MAXNAMELEN, DRMS_Env_struct::series_cache, and DRMS_Env_struct::session.

8.7 base/drms/libs/api/drms_segment.h File Reference

8.7.1 Detailed Description

Functions to access DRMS segment data structures.

See also:

drms_record.h drms_keyword.h drms_link.h drms_array.h

Definition in file drms_segment.h.

```
#include "drms_types.h"
```

Defines

- #define name2seg(rec, name) drms_segment_lookup(rec, name)
- #define num2seg(rec, num) drms_segment_lookupnum(rec, num)

Functions

- int **drms_segment_export** (DRMS_Segment_t *seg)
- int **drms_segment_export_ext** (DRMS_Segment_t *seg, DRMS_KeyMapClass_t *class, DRMS_KeyMap_t *map)

Lookup

- DRMS_Segment_t * drms_segment_lookup (DRMS_Record_t *record, const char *segname)
- DRMS_Segment_t * drms_segment_lookupnum (DRMS_Record_t *record, int segnum)

Create and Destroy

- HContainer_t * drms_create_segment_prototypes (DRMS_Record_t *target, DRMS_Record_t *source, int *status)
- HContainer_t * drms_segment_createinfocon (DRMS_Env_t *drmsEnv, const char *seriesName, int *status)
- void drms_segment_destroyinfocon (HContainer_t **info)

Information and Diagnostics

- void drms_segment_print (DRMS_Segment_t *seg)
- void drms_segment_filename (DRMS_Segment_t *seg, char *filename)
- long long drms_segment_size (DRMS_Segment_t *seg, int *status)
- int drms_segment_segsmatch (const DRMS_Segment_t *s1, const DRMS_Segment_t *s2)

Manipulate Axes

- int drms_segment_setdims (DRMS_Segment_t *seg, DRMS_SegmentDimInfo_t *di)
- int drms_segment_getdims (DRMS_Segment_t *seg, DRMS_SegmentDimInfo_t *di)

Scaling and Blocksize

- int drms_segment_setscaling (DRMS_Segment_t *seg, double bzero, double bscale)

- int drms_segment_getscaling (DRMS_Segment_t *seg, double *bzero, double *bscale)
- void drms_segment_autoscale (DRMS_Segment_t *seg, DRMS_Array_t *array)
- void drms_segment_setblocksize (DRMS_Segment_t *seg, int *blksz)
- void drms_segment_getblocksize (DRMS_Segment_t *seg, int *blksz)

Read and Write

- DRMS_Array_t * drms_segment_read (DRMS_Segment_t *seg, DRMS_Type_t type, int *status)
- DRMS_Array_t * drms_segment_readslice (DRMS_Segment_t *seg, DRMS_Type_t type, int *start, int *end, int *status)
- int drms_segment_write (DRMS_Segment_t *seg, DRMS_Array_t *arr, int autoscale)
- int drms_segment_write_from_file (DRMS_Segment_t *seg, char *infile)

8.7.2 Define Documentation

8.7.2.1 #define name2seg(rec, name) drms_segment_lookup(rec, name)

Synonym for drms_segment_lookup.

Definition at line 25 of file drms_segment.h.

8.7.2.2 #define num2seg(rec, num) drms_segment_lookupnum(rec, num)

Synonym for drms_segment_lookupnum.

Definition at line 29 of file drms_segment.h.

8.7.3 Function Documentation

8.7.3.1 HContainer_t* drms_create_segment_prototypes (DRMS_Record_t * *target*, DRMS_Record_t * *source*, int * *status*)

A DRMS segment prototype is a DRMS segment to which a DRMS record prototype is linked. Creates DRMS segment prototypes (type DRMS_Segment_t) for the target record prototype (*target*), using *source* as a template. The main difference between the DRMS segments in *source* and those in *target* is that *source* could be a record that is a series record template (`DRMS_Env_t.series_cache`) or a member of the record cache (`DRMS_Env_t.record_cache`), but *target* cannot be either of those types of records.

Parameters:

target Output DRMS record prototype which will contain the created DRMS segment prototypes.

source Input DRMS record, which contains source DRMS segments that will be used to initialize the created DRMS segment prototypes.

Returns:

Container pointing to the segment container of *target*.

Definition at line 145 of file drms_segment.c.

References DRMS_Segment_struct::axis, DRMS_Segment_struct::blocksize, DRMS_ERROR_OUTOFMEMORY, DRMS_MAXRANK, DRMS_TAS, DRMS_Segment_struct::info, DRMS_SegmentInfo_struct::name, HContainer_struct::num_total, DRMS_SegmentInfo_struct::protocol, DRMS_Segment_struct::record, and DRMS_Record_struct::segments.

8.7.3.2 void drms_segment_autoscale (DRMS_Segment_t * *seg*, DRMS_Array_t * *array*)

Sets the scaling parameters in the `seg->record` keyword (see drms_segment_setscaling), to values based on the extrema of the data *array*. The scaling parameters are set so that the unscaled data would occupy the full range of the segment fixed-point type; for example the maximum value of the array would be represented by 127 and the minimum by -128 in the segment data file if the type were DRMS_TYPE_CHAR. The scaling parameters will only be changed, however, if the original data would not otherwise fit unscaled into the range of the segment data type. Setting the scaling parameters for segments of floating-point type is probably a bad idea, but is done anyway, without regard to range. For other types, the scaling parameters are simply set to 1 and 0.

Parameters:

seg The segment whose scaling is to be retrieved.

array The DRMS array whose data is to be autoscaled.

Definition at line 1548 of file drms_segment.c.

References DRMS_Array_struct::bscale, DRMS_Array_struct::bzero, DRMS_Array_struct::data, drms_array_count(), DRMS_TYPE_CHAR, DRMS_TYPE_DOUBLE, DRMS_TYPE_FLOAT, DRMS_TYPE_INT, DRMS_TYPE_LONGLONG, DRMS_TYPE_SHORT, DRMS_TYPE_STRING, DRMS_TYPE_TIME, DRMS_Segment_struct::info, DRMS_Array_struct::israw, DRMS_Array_struct::type, and DRMS_SegmentInfo_struct::type.

8.7.3.3 HContainer_t* drms_segment_createinfocon (DRMS_Env_t * *drmsEnv*, const char * *seriesName*, int * *status*)

Creates an HContainer_t which contains DRMS_SegmentInfo_t structs. There is one such struct for each DRMS template segment (which is a template for series *seriesName*), with the former being a copy of the latter. The purpose of this function is to provide the user a container with segment information about pertinent to all of *seriesName's* segments.

Parameters:

drmsEnv DRMS session information.

seriesName The name of the series whose segment information is being copied.

status DRMS status (see drms_statuscodes.h). 0 if successful, non-0 otherwise.

Returns:

A created HContainer_t which contains information about all of series *seriesName's* segments. Caller is responsible for freeing the created HContainer_t by calling drms_segment_destroyinfocon.

Definition at line 499 of file drms_segment.c.

References DRMS_ERROR_OUTOFMEMORY, DRMS_MAXNAMELEN, DRMS_Segment_struct::info, and DRMS_SegmentInfo_struct::name.

8.7.3.4 void drms_segment_destroyinfocon (HContainer_t ** *info*)

Frees all memory allocated in the creation of *info*. To be used ONLY on HContainer_t structures created with drms_segment_createinfocon.

Parameters:

info Pointer to a pointer to the HContainer_t struct that contains series-specific segment information.

Definition at line 559 of file drms_segment.c.

8.7.3.5 void drms_segment_filename (DRMS_Segment_t * *seg*, char * *filename*)

Returns the absolute path to the segment file associated with *seg* in *filename*. The size of the buffer to which *filename* points must be at least DRMS_MAXPATHLEN+1 bytes long.

Parameters:

seg DRMS segment whose associated file's name will be returned.

filename Buffer to hold the filename upon return.

Definition at line 566 of file drms_segment.c.

References DRMS_DSDS, DRMS_LOCAL, DRMS_TAS, DRMS_Segment_struct::filename, DRMS_Segment_struct::info, DRMS_SegmentInfo_struct::name, DRMS_SegmentInfo_struct::protocol, DRMS_Segment_struct::record, DRMS_Record_struct::slotnum, and DRMS_Record_struct::su.

8.7.3.6 void drms_segment_getblocksize (DRMS_Segment_t * *seg*, int * *blksz*)

Copies the `seg->blocksize` array into the array *blksz*. *blksz* must be dimensioned to the rank of the segment. The blocksize is reserved for use with segments of protocol DRMS_TAS (tiled array storage) only, but is not used in any of the read/write functions described here.

Parameters:

seg The segment whose blocksize array is to be set.

blksz Input blocksizes.

Definition at line 1539 of file drms_segment.c.

References DRMS_Segment_struct::blocksize, DRMS_Segment_struct::info, and DRMS_SegmentInfo_struct::naxis.

8.7.3.7 int drms_segment_getdims (DRMS_Segment_t * *seg*, DRMS_SegmentDimInfo_t * *di*)

Sets the values of the *di* struct to the rank and axis lengths of *seg*.

Parameters:

seg DRMS segment whose information is used to initialize *di*.

di Receives, by reference, the segment information in *seg*.

Returns:

DRMS status (see drms_statuscodes.h). 0 if successful, non-0 otherwise.

Definition at line 486 of file drms_segment.c.

References DRMS_Segment_struct::axis, DRMS_SegmentDimInfo_struct::axis, DRMS_Segment_struct::info, DRMS_SegmentInfo_struct::naxis, and DRMS_SegmentDimInfo_struct::naxis.

8.7.3.8 int drms_segment_getscaling (DRMS_Segment_t * *seg*, double * *bzero*, double * *bscale*)

Returns the scaling parameter keywords, if they are present in the record struct `seg->record` (see drms_segment_setscaling) as **bscale* and **bzero*. If no valid values are found, the values 1.0 and 0,0, respectively, are returned.

Parameters:

seg The segment whose scaling is to be retrieved.

bzero Offset by which raw data values are to be shifted to produce actual data values, returned by reference.

bscale Scaling factor by which raw data values are to be multiplied to produce actual data value, returned by reference.

Returns:

DRMS status (see drms_statuscodes.h). 0 if successful, non-0 otherwise.

Definition at line 632 of file drms_segment.c.

References DRMS_Segment_struct::info, DRMS_Segment_struct::record, and DRMS_SegmentInfo_-struct::segnum.

8.7.3.9 DRMS_Segment_t* drms_segment_lookup (DRMS_Record_t * *record*, const char * *segname*)

Returns the segment associated with the name *segname* in record *record*. If the segment refers to a constant segment that has not yet been set, then the segment for the current record *record* is returned; otherwise, the constant segment is returned.

Parameters:

record The DRMS record that contains the DRMS segment being looked up.

segname The name of the DRMS segment being looked up.

Returns:

The DRMS segment specified by *record* and *segname*.

Definition at line 649 of file drms_segment.c.

References DRMS_SegmentInfo_struct::cseg_recnum, DRMS_CONSTANT, DRMS_Record_struct::env, DRMS_Segment_struct::info, DRMS_Record_struct::recnum, DRMS_Segment_struct::record, DRMS_-SegmentInfo_struct::scope, and DRMS_Record_struct::seriesinfo.

8.7.3.10 DRMS_Segment_t* drms_segment_lookupnum (DRMS_Record_t * *record*, int *segnum*)

Returns the segment associated with the number *segnum* in record *record*. For constant segments, it behaves the same as drms_segment_lookup.

Parameters:

record The DRMS record that contains the DRMS segment being looked up.

segnum The number of the DRMS segment being looked up.

Returns:

The DRMS segment specified by *record* and *segnum*.

Definition at line 676 of file drms_segment.c.

References DRMS_Segment_struct::info, DRMS_SegmentInfo_struct::name, and DRMS_Record_-struct::segments.

8.7.3.11 void drms_segment_print (DRMS_Segment_t * *seg*)

Prints the full *seg* struct information to stdout.

Parameters:

seg DRMS segment struct whose fields will be printed to stdout.

Definition at line 360 of file drms_segment.c.

References DRMS_Segment_struct::axis, DRMS_Segment_struct::blocksize, DRMS_SegmentInfo_-struct::description, DRMS_BINARY, DRMS_BINZIP, DRMS_CONSTANT, DRMS_FITS, DRMS_-FITZ, DRMS_GENERIC, DRMS_MSI, DRMS_TAS, DRMS_VARDIM, DRMS_VARIABLE, DRMS_-Segment_struct::filename, DRMS_Segment_struct::info, DRMS_SegmentInfo_struct::islink, DRMS_-SegmentInfo_struct::linkname, DRMS_SegmentInfo_struct::name, DRMS_SegmentInfo_struct::naxis, DRMS_SegmentInfo_struct::protocol, DRMS_SegmentInfo_struct::scope, DRMS_SegmentInfo_-struct::segnum, DRMS_SegmentInfo_struct::target_seg, DRMS_SegmentInfo_struct::type, and DRMS_-SegmentInfo_struct::unit.

8.7.3.12 DRMS_Array_t* drms_segment_read (DRMS_Segment_t * *seg*, DRMS_Type_t *type*, int * *status*)

Reads the data associated with *seg* into memory in a newly created DRMS_Array_t struct, converting the data to the requested *type* (unless *type* = DRMS_TYPE_RAW, in which case the data type will be that of the external representation.).

Parameters:

seg The segment whose file is to be read into memory.

type The type to which the data of is converted.

status DRMS status (see drms_statuscodes.h). 0 if successful, non-0 otherwise.

Returns:

The created DRMS array struct.

Definition at line 703 of file drms_segment.c.

References DRMS_Segment_struct::axis, DRMS_Array_struct::axis, DRMS_Array_struct::bscale, DRMS_Array_struct::bzero, DRMS_SegmentInfo_struct::cseg_recnum, DRMS_Array_struct::data, drms_array_size(), DRMS_BINARY, DRMS_BINZIP, DRMS_CONSTANT, DRMS_DSDS, DRMS_-ERROR_OUTOFMEMORY, DRMS_FITS, DRMS_FITZ, DRMS_GENERIC, DRMS_LOCAL, DRMS_MAXRANK, DRMS_MSI, DRMS_TAS, DRMS_TYPE_RAW, DRMS_Record_struct::env, DRMS_Segment_struct::filename, DRMS_Segment_struct::info, DRMS_Array_struct::israw, DRMS_-SegmentInfo_struct::naxis, DRMS_Array_struct::naxis, DRMS_Array_struct::parent_segment, DRMS_-SegmentInfo_struct::protocol, DRMS_Segment_struct::record, DRMS_SegmentInfo_struct::scope,

DRMS_Record_struct::seriesinfo, DRMS_Record_struct::slotnum, DRMS_Array_struct::start, DRMS_-Record_struct::su, DRMS_Record_struct::sunum, DRMS_SegmentInfo_struct::type, and DRMS_Array_-struct::type.

8.7.3.13 DRMS_Array_t* drms_segment_readslice (DRMS_Segment_t * *seg*, DRMS_Type_t *type*, int * *start*, int * *end*, int * *status*)

Similar to drms_segment_read, except that only the data between the *start*[n] and *end*[n] values in each dimension *n* are read into the array. *start* and *end* must be vectors of rank equal to that of the data segment.

Parameters:

seg The segment whose file slice is to be read into memory.

type The type to which the data of is converted.

start The index value, in all dimensions, that starts the slice.

end The index value, in all dimensions, that ends the slice.

Returns:

The created DRMS array struct.

Definition at line 1088 of file drms_segment.c.

References DRMS_Segment_struct::axis, DRMS_Array_struct::axis, DRMS_Array_struct::bscale, DRMS_Array_struct::bzero, DRMS_BINARY, DRMS_BINZIP, DRMS_FITS, DRMS_FITZ, DRMS_-GENERIC, DRMS_MAXRANK, DRMS_MSI, DRMS_TAS, DRMS_TYPE_RAW, DRMS_Record_-struct::env, DRMS_Segment_struct::info, DRMS_Array_struct::israw, DRMS_Array_struct::naxis, DRMS_SegmentInfo_struct::naxis, DRMS_Array_struct::parent_segment, DRMS_SegmentInfo_-struct::protocol, DRMS_Segment_struct::record, DRMS_Record_struct::seriesinfo, DRMS_Record_-struct::slotnum, DRMS_Array_struct::start, DRMS_Record_struct::su, DRMS_Record_struct::sunum, DRMS_SegmentInfo_struct::type, and DRMS_Array_struct::type.

8.7.3.14 int drms_segment_segsmatch (const DRMS_Segment_t * *s1*, const DRMS_Segment_t * *s2*)

Returns 1 if both segments exist (or are NULL) and have the same rank, dimensions, protocol, blocksizes (if they are of protocol DRMS_TAS), type, and scope; 0 otherwise.

Parameters:

s1 DRMS segment whose information is being compared.

s2 DRMS segment whose information is being compared.

Returns:

1 if *s1* and *s2* match; 0 otherwise.

Definition at line 1841 of file drms_segment.c.

References DRMS_Segment_struct::axis, DRMS_Segment_struct::blocksize, DRMS_TAS, DRMS_-Segment_struct::info, DRMS_SegmentInfo_struct::naxis, DRMS_SegmentInfo_struct::protocol, DRMS_-SegmentInfo_struct::scope, and DRMS_SegmentInfo_struct::type.

8.7.3.15 void drms_segment_setblocksize (DRMS_Segment_t * *seg*, int * *blksz*)

Sets the `seg->blocksize` array to the array of blocksizes *blksz*.

Parameters:

seg The segment whose blocksize array is to be set.

blksz Input blocksizes.

Definition at line 1533 of file drms_segment.c.

References DRMS_Segment_struct::blocksize, DRMS_Segment_struct::info, and DRMS_SegmentInfo_struct::naxis.

8.7.3.16 int drms_segment_setdims (DRMS_Segment_t * *seg*, DRMS_SegmentDimInfo_t * *di*)

Sets the rank and axis lengths of *seg* to those of the *di* struct.

Parameters:

seg DRMS segment struct whose fields will be set.

di Segment information used to initialized *seg*.

Returns:

DRMS status (see drms_statuscodes.h). 0 if successful, non-0 otherwise.

Definition at line 468 of file drms_segment.c.

References DRMS_SegmentDimInfo_struct::axis, DRMS_Segment_struct::axis, DRMS_ERROR_INVALIDDATA, DRMS_Segment_struct::info, DRMS_SegmentDimInfo_struct::naxis, and DRMS_SegmentInfo_struct::naxis.

8.7.3.17 int drms_segment_setscaling (DRMS_Segment_t * *seg*, double *bzero*, double *bscale*)

Sets the values for key (metadata) values `bscale[n]` and `bzero[n]` in the `seg->record` struct to *bscale* and *bzero* respectively, where *n* is the segment number, `seg->info->segnum`. These values are used by drms_segment_read and drms_segment_write functions to scale the externally represented data.

Parameters:

seg The segment whose data is to be scaled.

bzero Offset by which raw data values are to be shifted to produce actual data values.

bscale Scaling factor by which raw data values are to be multiplied to produce actual data values.

Returns:

DRMS status (see drms_statuscodes.h). 0 if successful, non-0 otherwise.

Definition at line 610 of file drms_segment.c.

References DRMS_Segment_struct::info, DRMS_Record_struct::readonly, DRMS_Segment_struct::record, and DRMS_SegmentInfo_struct::segnum.

8.7.3.18 long long drms_segment_size (DRMS_Segment_t * *seg*, int * *status*)

Returns the total size of the *seg* data array, in bytes (product of the number of data elements and size of the datatype). If the segment data type is DRMS_TYPE_STRING, the size returned is only the number of data elements times the size of an address. So, for the DRMS_TYPE_STRING datatype, this function probably does not provide the desired information.

Parameters:

seg DRMS segment whose data size is being estimated.

status DRMS status (see drms_statuscodes.h). 0 if successful, non-0 otherwise.

Returns:

The esimated data size, in bytes.

Definition at line 451 of file drms_segment.c.

References DRMS_Segment_struct::axis, DRMS_TYPE_STRING, DRMS_Segment_struct::info, DRMS_SegmentInfo_struct::naxis, and DRMS_SegmentInfo_struct::type.

8.7.3.19 int drms_segment_write (DRMS_Segment_t * *seg*, DRMS_Array_t * *arr*, int *autoscale*)

Writes the data from the DRMS array *arr* into the file associated with the segment *seg*, provided that the segment uses one of the supported non-ready-only protocols (DRMS_BINARY, DRMS_BINZIP, DRMS_FITS, DRMS_FITZ, and DRMS_TAS). The array dimensions must match those of the segment. If *autoscale* is non-zero, the function drms_segment_autoscale is invoked before output. In any case, conversion to the data type and scaling appropriate to the segment is performed.

Parameters:

seg The segment whose file is to be written to the filesystem.

arr The array containing the data that is to be written to the output file.

autoscale If 0, do not invoke drms_segment_autoscale. Otherwise, invoke drms_segment_autoscale.

Returns:

DRMS status (see drms_statuscodes.h). 0 if successful, non-0 otherwise.

Definition at line 1276 of file drms_segment.c.

References DRMS_Segment_struct::axis, DRMS_Array_struct::axis, DRMS_Array_struct::bscale, DRMS_Array_struct::bzero, DRMS_SegmentInfo_struct::cseg_recnum, DRMS_Array_struct::data, DRMS_BINARY, DRMS_BINZIP, DRMS_CONSTANT, DRMS_FITS, DRMS_FITZ, DRMS_GENERIC, DRMS_MAXRANK, DRMS_MAXSEGFILENAME, DRMS_MSI, DRMS_TAS, DRMS_Segment_struct::filename, DRMS_Segment_struct::info, DRMS_Array_struct::israw, DRMS_Record_struct::lifetime, DRMS_SegmentInfo_struct::naxis, DRMS_Array_struct::naxis, DRMS_SegmentInfo_struct::protocol, DRMS_Record_struct::readonly, DRMS_Segment_struct::record, DRMS_SegmentInfo_struct::scope, DRMS_Record_struct::seriesinfo, DRMS_Record_struct::slotnum, DRMS_SegmentInfo_struct::type, and DRMS_Array_struct::type.

8.7.3.20 int drms_segment_write_from_file (DRMS_Segment_t * *seg*, char * *infile*)

Simply copies the contents of the file specified by *infile* into the file associated with *seg*. It can only be used for segments whose protocol is DRMS_GENERIC.

Parameters:

seg The segment whose file is to be written to the filesystem.

infile The input file whose contents are to be written to the file owned by .

Returns:

DRMS status (see drms_statuscodes.h). 0 if successful, non-0 otherwise.

Definition at line 1450 of file drms_segment.c.

References DRMS_SegmentInfo_struct::cseg_recnum, DRMS_CONSTANT, DRMS_GENERIC, DRMS_MAXSEGFILENAME, DRMS_Segment_struct::filename, DRMS_Segment_struct::info, DRMS_Record_struct::lifetime, DRMS_SegmentInfo_struct::protocol, DRMS_Record_struct::readonly, DRMS_Segment_struct::record, DRMS_SegmentInfo_struct::scope, DRMS_Record_struct::seriesinfo, DRMS_Record_struct::slotnum, and DRMS_Record_struct::su.

8.8 base/drms/libs/api/drms_segment_priv.h File Reference

8.8.1 Detailed Description

Definition in file drms_segment_priv.h.

```
#include "drms_types.h"
```

Functions

Create and Destroy

- int drms_template_segments (DRMS_Record_t *template)
- int drms_delete_segmentfile (DRMS_Segment_t *seg)
- void drms_free_segment_struct (DRMS_Segment_t *segment)
- void drms_free_template_segment_struct (DRMS_Segment_t *segment)

Replicate

- void drms_copy_segment_struct (DRMS_Segment_t *dst, DRMS_Segment_t *src)

8.8.2 Function Documentation

8.8.2.1 void drms_copy_segment_struct (DRMS_Segment_t * *dst*, DRMS_Segment_t * *src*)

Copies the entire *src* DRMS segment struct to *dst*. NOTE: *src* will retain pointers to the allocated members of *src* (ie., `src->record` and `src->info`). This implies that BOTH *dst* and *src* will have some pointers in common, so care must be ensured when freeing memory.

Parameters:

dst DRMS segment struct whose fields will be initialized by the fields of *src*.

src DRMS segment struct whose fields will be copied.

Definition at line 137 of file drms_segment.c.

8.8.2.2 int drms_delete_segmentfile (DRMS_Segment_t * *seg*)

Removes the file storing the data associated with *seg*.

Parameters:

seg DRMS segment whose associated data file will be deleted.

Returns:

DRMS status (see drms_statuscodes.h). 0 if successful, non-0 otherwise.

Definition at line 599 of file drms_segment.c.

8.8.2.3 void drms_free_segment_struct (DRMS_Segment_t * *segment*)

Shallow-free a DRMS segment structure. For some reason, this is a no-op, unless *segment* is NULL, in which case an error is generated.

Parameters:

segment The DRMS segment to free.

Definition at line 132 of file drms_segment.c.

8.8.2.4 void drms_free_template_segment_struct (DRMS_Segment_t * *segment*)

Deep-free a DRMS segment structure. This function does NOT free the actual DRMS_Segment_t. It only frees the allocated memory to which it has pointers.

Parameters:

segment The DRMS segment to free.

Definition at line 128 of file drms_segment.c.

References DRMS_Segment_struct::info.

8.8.2.5 int drms_template_segments (DRMS_Record_t * *template*)

Builds the segment part of a dataseries record template from the record *template* by querying the database and using ther results to initialize the array of segment descriptors.

Parameters:

template DRMS dataseries record template (which resides in `DRMS_Env_t::series_cache`)

Returns:

DRMS status (see drms_statuscodes.h). 0 if successful, non-0 otherwise.

Definition at line 223 of file drms_segment.c.

References DRMS_Segment_struct::axis, DRMS_Segment_struct::blocksize, DRMS_SegmentInfo_struct::cseg_recnum, DRMS_SegmentInfo_struct::description, DRMS_CONSTANT, DRMS_GENERIC, DRMS_MAXCOMMENTLEN, DRMS_MAXNAMELEN, DRMS_MAXUNITLEN, DRMS_TAS, DRMS_TYPE_INT, DRMS_VARDIM, DRMS_VARIABLE, DRMS_Segment_struct::filename, DRMS_Segment_struct::info, DRMS_SegmentInfo_struct::islink, DRMS_SegmentInfo_struct::linkname, DRMS_SegmentInfo_struct::name, DRMS_SegmentInfo_struct::naxis, DRMS_SegmentInfo_struct::protocol, DRMS_Segment_struct::record, DRMS_SegmentInfo_struct::scope, DRMS_SegmentInfo_struct::segnum, DRMS_Env_struct::session, DRMS_SegmentInfo_struct::target_seg, DRMS_SegmentInfo_struct::type, and DRMS_SegmentInfo_struct::unit.

8.9 base/drms/libs/api/drms_series.h File Reference

8.9.1 Detailed Description

Functions to query the existence of series, get information about a series' primary keywords, and check series compatibility.

See also:

drms_keyword.h drms_record.h drms_segment.h drms_statuscodes.h drms_types.h hcontainer.h

Definition in file drms_series.h.

```
#include "drms_types.h"
```

Functions

- int drms_series_exists (DRMS_Env_t *drmsEnv, const char *sname, int *status)
- int drms_insert_series (DRMS_Session_t *session, int update, DRMS_Record_t *template, int perms)

  insert a DRMS series

- char ** drms_series_createpkeyarray (DRMS_Env_t *env, const char *seriesName, int *nPKeys, int *status)
- void drms_series_destroypkeyarray (char ***pkeys, int nElements)
- int drms_series_checkseriescompat (DRMS_Env_t *drmsEnv, const char *series1, const char *series2, HContainer_t *matchSegs, int *status)
- int drms_series_checkrecordcompat (DRMS_Env_t *drmsEnv, const char *series, DRMS_Record_t *recTempl, HContainer_t *matchSegs, int *status)
- int drms_series_checkkeycompat (DRMS_Env_t *drmsEnv, const char *series, DRMS_Keyword_t *keys, int nKeys, int *status)
- int drms_series_checksegcompat (DRMS_Env_t *drmsEnv, const char *series, DRMS_Segment_t *segs, int nSegs, int *status)

8.9.2 Function Documentation

8.9.2.1 int drms_series_checkkeycompat (DRMS_Env_t * *drmsEnv*, const char * *series*, DRMS_Keyword_t * *keys*, int *nKeys*, int * *status*)

queries the DRMS database to determine if each of the keywords in *keys*, an array of DRMS_Keyword_t, "match" a keyword defined for series *series*. drms_keyword_keysmatch is used to determined if the two DRMS_Keyword_t match. The caller must provide, in the *nKeys* parameter, the number of keywords contained in *keys*.

Returns:

1 if compatible. Upon success, this function sets **status* to DRMS_SUCCESS. Otherwise **status* is set to a diagostic status code (see drms_statuscodes.h).

Examples:

drms_series_ex5.c.

Definition at line 976 of file drms_series.c.

References DRMS_Keyword_struct::info.

8.9.2.2 int drms_series_checkrecordcompat (DRMS_Env_t * *drmsEnv*, const char * *series*, DRMS_Record_t * *recTempl*, HContainer_t * *matchSegs*, int * *status*)

queries the DRMS database to determine if *recTempl* is compatible with the series named *series* in terms of their primary-keyword and segment definitions. The two are compatible if an attempt to insert *recTempl* into the series named *series* would succeed. This function should be used whenever a record is created by a means other than drms_create_records or drms_clone_records, and the user will attempt to insert that newly created record into series *series*.

Returns:

1 if compatible. Upon success, this function sets ∗*status* to DRMS_SUCCESS. Otherwise ∗*status* is set to a diagostic status code (see drms_statuscodes.h).

Examples:

drms_series_ex3.c.

Definition at line 897 of file drms_series.c.

References DRMS_Record_struct::seriesinfo.

8.9.2.3 int drms_series_checksegcompat (DRMS_Env_t * *drmsEnv*, const char * *series*, DRMS_Segment_t * *segs*, int *nSegs*, int * *status*)

queries the DRMS database to determine if each of the segments in *segs*, an array of DRMS_Segment_t, "match" a segment defined for series *series*. drms_segment_segsmatch is used to determine if two segment definitions match. The caller must provide, in the *nSegs* parameter, the number of segments contained in *segs*.

Returns:

1 if compatible. Upon success, this function sets ∗*status* to DRMS_SUCCESS. Otherwise ∗*status* is set to a diagostic status code (see drms_statuscodes.h).

Examples:

drms_series_ex4.c.

Definition at line 1015 of file drms_series.c.

References DRMS_Segment_struct::info, and DRMS_SegmentInfo_struct::name.

8.9.2.4 int drms_series_checkseriescompat (DRMS_Env_t * *drmsEnv*, const char * *series1*, const char * *series2*, HContainer_t * *matchSegs*, int * *status*)

queries the DRMS database to determine if *series1* and *series2* are compatible in terms of their primary-keyword and segment definitions. Two primary keyword definitions "match" if the type, isconstant, and per_segment DRMS_Keyword_t fields are equal. drms_segment_segsmatch is used to determine if two segment definitions "match". If all primary keywords and at least one segment match and no error occurs,

the function intializes *matchSegs* and allocates one HContainer_t element for each segment that the two series have in common. Each HContainer_t element is a string that names the matching segment. The caller must provide a non-NULL *matchSegs* container to the function. It is the caller's responsibility to deallocate memory associated with the allocation of HContainer_t elements.

Returns:

1 if compatible. Upon success, this function sets ∗*status* to DRMS_SUCCESS. Otherwise ∗*status* is set to a diagostic status code (see drms_statuscodes.h).

Definition at line 859 of file drms_series.c.

8.9.2.5 char∗∗ drms_series_createpkeyarray (DRMS_Env_t ∗ *env*, const char ∗ *seriesName*, int ∗ *nPKeys*, int ∗ *status*)

allocates and returns an array of null-terminated strings, each of which is the name of a primary keyword of the series *seriesName*. It is the caller's responsibility to free the allocated array by calling drms_series_destroypkeyarray. Upon success, drms_series_createpkeyarray returns by reference in the *nPKeys* parameter the number of primary keywords that belong to the series *seriesName*.

Returns:

a primary key array upon success.
∗*status* is set to DRMS_SUCCESS upon success. Otherwise ∗*status* is set to a diagostic status code (see drms_statuscodes.h), which includes the following:

- DRMS_ERROR_UNKNOWNSERIES
- DRMS_ERROR_OUTOFMEMORY
- DRMS_ERROR_INVALIDDATA

Examples:

drms_series_ex2.c.

Definition at line 822 of file drms_series.c.

8.9.2.6 void drms_series_destroypkeyarray (char ∗∗∗ *pkeys*, int *nElements*)

deallocates an array of strings allocated by drms_series_createpkeyarray. The user must specify the number of primary keywords in the *nElements* parameter. Failure to provide the correct number could result in undetermined program behavior.

Upon success, this function sets ∗*pkeys* to NULL.

Examples:

drms_series_ex2.c.

Definition at line 839 of file drms_series.c.

8.9.2.7 int drms_series_exists (DRMS_Env_t ∗ *drmsEnv*, const char ∗ *sname*, int ∗ *status*)

queries the DRMS database to determine if the series specified by *sname* exists.

Returns:

If a null value for *sname*, or an empty string, is provided, the function returns 0. Otherwise, if *sname* is a valid series name, but the series does not exist in DRMS, the function returns 0, and sets ∗*status* to DRMS_ERROR_UNKNOWNSERIES. If there is an error looking up *sname*, the function returns 0, and sets ∗*status* to the value that would result by calling drms_template_record with *sname*. If *sname* is a valid series name, and it names an existing DRMS series, the function returns 1.

Examples:

drms_series_ex1.c.

Definition at line 28 of file drms_series.c.

References DRMS_ERROR_UNKNOWNSERIES.

8.10 base/drms/libs/api/drms_statuscodes.h File Reference

8.10.1 Detailed Description

Definition in file drms_statuscodes.h.

Defines

- #define **CHECKNULL**(ptr) if (!(ptr)) return DRMS_ERROR_NULLPOINTER
- #define **CHECKNULL_STAT**(ptr, stat)
- #define **CHECKSNPRINTF**(code, len)
- #define **DRMS_NO_ERROR** (0)
- #define **DRMS_SUCCESS** (0)
- #define **DRMS_VALUE_MISSING** (-3)
- #define **DRMS_BADSTRING** (-2)
- #define **DRMS_RANGE** (-1)
- #define **DRMS_EXACT** (0)
- #define **DRMS_INEXACT** (1)
- #define **DRMS_ERROR_BADSEQUENCE** (-10001)
- #define **DRMS_ERROR_BADTEMPLATE** (-10002)
- #define DRMS_ERROR_UNKNOWNSERIES (-10003)

  the parameter specifying the series name refers to a non-existent series

- #define **DRMS_ERROR_UNKNOWNRECORD** (-10004)
- #define **DRMS_ERROR_UNKNOWNLINK** (-10005)
- #define **DRMS_ERROR_UNKNOWNKEYWORD** (-10006)
- #define **DRMS_ERROR_UNKNOWNSEGMENT** (-10007)
- #define **DRMS_ERROR_BADFIELDCOUNT** (-10008)
- #define **DRMS_ERROR_INVALIDLINKTYPE** (-10009)
- #define **DRMS_ERROR_BADLINK** (-10010)
- #define **DRMS_ERROR_UNKNOWNUNIT** (-10011)
- #define **DRMS_ERROR_QUERYFAILED** (-10012)
- #define **DRMS_ERROR_BADQUERYRESULT** (-10013)
- #define **DRMS_ERROR_UNKNOWNSU** (-10014)
- #define **DRMS_ERROR_RECORDREADONLY** (-10015)
- #define **DRMS_ERROR_KEYWORDREADONLY** (-10016)
- #define **DRMS_ERROR_NOTIMPLEMENTED** (-10017)
- #define **DRMS_ERROR_UNKNOWNPROTOCOL** (-10018)
- #define **DRMS_ERROR_NULLPOINTER** (-10019)
- #define **DRMS_ERROR_INVALIDTYPE** (-10020)
- #define **DRMS_ERROR_INVALIDDIMS** (-10021)
- #define **DRMS_ERROR_INVALIDACTION** (-10022)
- #define **DRMS_ERROR_COMMITREADONLY** (-10023)
- #define **DRMS_ERROR_SYNTAXERROR** (-10024)
- #define **DRMS_ERROR_BADRECORDCOUNT** (-10025)
- #define **DRMS_ERROR_NULLENV** (-10026)
- #define DRMS_ERROR_OUTOFMEMORY (-10027)

  a memory allocation fails due to insufficient memory available

- #define **DRMS_ERROR_UNKNOWNCOMPMETH** (-10028)
- #define **DRMS_ERROR_COMPRESSFAILED** (-10029)
- #define **DRMS_ERROR_INVALIDRANK** (-10030)
- #define **DRMS_ERROR_MKDIRFAILED** (-10031)
- #define **DRMS_ERROR_UNLINKFAILED** (-10032)
- #define **DRMS_ERROR_STATFAILED** (-10033)
- #define **DRMS_ERROR_SUMOPEN** (-10034)
- #define **DRMS_ERROR_SUMPUT** (-10035)
- #define **DRMS_ERROR_SUMGET** (-10036)
- #define **DRMS_ERROR_SUMALLOC** (-10037)
- #define **DRMS_ERROR_SUMWAIT** (-10038)
- #define **DRMS_ERROR_SUMBADOPCODE** (-10039)
- #define **DRMS_ERROR_INVALIDFILE** (-10040)
- #define **DRMS_ERROR_IOERROR** (-10041)
- #define **DRMS_ERROR_LINKNOTSET** (-10042)
- #define **DRMS_ERROR_BADJSD** (-10043)
- #define **DRMS_ERROR_INVALIDRECORD** (-10044)
- #define **DRMS_ERROR_INVALIDKEYWORD** (-10045)
- #define **DRMS_ERROR_INVALIDSEGMENT** (-10046)
- #define **DRMS_ERROR_INVALIDLINK** (-10047)
- #define DRMS_ERROR_INVALIDDATA (-10048)

  a parameter to a function has an unexpected value (such as providing a null pointer when a character string is expected)

- #define **DRMS_ERROR_NODSDSSUPPORT** (-10049)
- #define **DRMS_ERROR_LIBDSDS** (-10050)
- #define **DRMS_ERROR_ABORT** (-10051)
- #define **DRMS_ERROR_CANTOPENLIBRARY** (-10052)
- #define **DRMS_WARNING_BADBLANK** (10000)
- #define **DRMS_QUERY_TRUNCATED** (10001)

8.10.2 Define Documentation

8.10.2.1 #define CHECKNULL_STAT(ptr, stat)

Value:

```
do {  \
                                        if (!(ptr)) { \
                                          if ((stat)) \
                                            *(stat) = DRMS_ERROR_NULLPOINTER; \
                                            fprintf(stderr,"ERROR at %s, line %d: "#ptr" = NULL.\n",__FILE__
                                          return NULL; \
                                        } \
                                      } while(0)
```

Definition at line 9 of file drms_statuscodes.h.

8.10.2.2 #define CHECKSNPRINTF(code, len)

Value:

```
do {\
  if ((code) >= (len)) { \
    fprintf(stderr, "WARNING: string is truncated in %s, line %d\n",__FILE__,__LINE__); \
  }\
} while (0)
```

Definition at line 17 of file drms_statuscodes.h.

8.11 base/drms/libs/api/drms_types.h File Reference

8.11.1 Detailed Description

Definition in file drms_types.h.

```
#include <math.h>
#include <float.h>
#include <limits.h>
#include <stdint.h>
#include "db.h"
#include "hcontainer.h"
#include "util.h"
#include "tagfifo.h"
#include "drms_protocol.h"
```

Classes

- union DRMS_Type_Value
  DRMS type value.
- struct **DRMS_Value**
- struct DRMS_Session_struct
  DRMS Session struct.
- struct **DS_node_struct**
- struct DRMS_Env_struct
  DRMS environment struct.
- struct **DRMS_ThreadInfo_struct**
- struct **DRMS_SumRequest_struct**
- struct **DRMS_RecordSet_struct**
- struct **DRMS_SeriesInfo_struct**
- struct DRMS_Record_struct
  DRMS record struct.
- struct **DRMS_KeywordInfo_struct**
- struct DRMS_Keyword_struct
- struct **DRMS_LinkInfo_struct**
- struct DRMS_Link_struct
  DRMS link struct.
- struct DRMS_Array_struct
  DRMS array struct.
- struct DRMS_SegmentDimInfo_struct
  DRMS segment dimension info struct.

- struct DRMS_SegmentInfo_struct

  DRMS segment info struct.

- struct DRMS_Segment_struct

  DRMS segment struct.

- struct **DRMS_StorageUnit_struct**
- struct DRMS_KeyMap_struct

  DRMS keymap struct.

Defines

- #define DRMS_MAXNAMELEN (32)
- #define DRMS_MAXHASHKEYLEN (DRMS_MAXNAMELEN+22)

  Maximum DRMS hash byte length.

- #define DRMS_MAXUNITLEN (32)

  Maximum byte length of unit string.

- #define **DRMS_MAXQUERYLEN** (8192)
- #define **DRMS_MAXPATHLEN** (512)
- #define **DRMS_MAXFORMATLEN** (20)
- #define DRMS_MAXRANK (16)

  Maximum dimension of DRMS data.

- #define DRMS_MAXSEGMENTS (255)

  Maximum number of DRMS segments per record.

- #define DRMS_MAXCOMMENTLEN (255)

  Maximum DRMS comment byte length.

- #define DRMS_MAXSEGFILENAME (256)

  Maximum byte length of DRMS segment file name.

- #define **DRMS_MAXPRIMIDX** (5)
- #define **DRMS_DEFVAL_MAXLEN** (1000)
- #define **DRMS_MAXLINKDEPTH** (20)
- #define **DRMS_MAXHOSTNAME** (128)
- #define **F_NAN** (__f_nan__.val)
- #define **D_NAN** (__d_nan__.val)
- #define DRMS_MISSING_CHAR (SCHAR_MIN)

  DRMS char missing value.

- #define DRMS_MISSING_SHORT (SHRT_MIN)

  DRMS short missing value.

- #define DRMS_MISSING_INT (INT_MIN)

DRMS int missing value.

- #define DRMS_MISSING_LONGLONG (LLONG_MIN)

  DRMS longlong missing value.

- #define DRMS_MISSING_FLOAT (F_NAN)

  DRMS float missing value.

- #define DRMS_MISSING_DOUBLE (D_NAN)

  DRMS double missing value.

- #define DRMS_MISSING_STRING ("")

  DRMS C string missing value.

- #define DRMS_MISSING_TIME (-211087684800.0)

  DRMS time missing value.

- #define **DRMS_MAXTYPENAMELEN** (9)
- #define **DRMS_SUMALLOC** 0
- #define **DRMS_SUMGET** 1
- #define **DRMS_SUMPUT** 2
- #define **DRMS_SUMCLOSE** 3
- #define **DRMS_SUMABORT** 99
- #define **DRMS_MAX_REQCNT** 32
- #define **DRMS_READONLY** 1
- #define **DRMS_READWRITE** 2
- #define **DRMS_VAL_SET**(T, IV, OV)

Typedefs

- typedef union DRMS_Type_Value DRMS_Type_Value_t

  DRMS type value reference.

- typedef struct DRMS_Value **DRMS_Value_t**
- typedef struct DRMS_Session_struct DRMS_Session_t
- typedef struct DS_node_struct **DS_node_t**
- typedef struct DRMS_Env_struct DRMS_Env_t

  DRMS environment struct reference.

- typedef struct DRMS_ThreadInfo_struct **DRMS_ThreadInfo_t**
- typedef struct DRMS_SumRequest_struct **DRMS_SumRequest_t**
- typedef enum DRMS_RecordQueryType_struct **DRMS_RecordQueryType_t**
- typedef struct DRMS_RecordSet_struct **DRMS_RecordSet_t**
- typedef struct DRMS_SeriesInfo_struct **DRMS_SeriesInfo_t**
- typedef struct DRMS_Record_struct DRMS_Record_t
- typedef struct DRMS_KeywordInfo_struct **DRMS_KeywordInfo_t**
- typedef struct DRMS_Keyword_struct DRMS_Keyword_t

  DRMS keyword struct reference.

- typedef struct DRMS_LinkInfo_struct **DRMS_LinkInfo_t**

- typedef struct DRMS_Link_struct DRMS_Link_t

  DRMS link struct reference.

- typedef struct DRMS_Array_struct DRMS_Array_t

  DRMS array struct reference.

- typedef struct DRMS_SegmentDimInfo_struct DRMS_SegmentDimInfo_t

  DRMS segment dim info struct reference.

- typedef struct DRMS_SegmentInfo_struct DRMS_SegmentInfo_t

  DRMS segment info struct reference.

- typedef struct DRMS_Segment_struct DRMS_Segment_t

  DRMS segment struct reference.

- typedef struct DRMS_StorageUnit_struct **DRMS_StorageUnit_t**
- typedef enum DRMS_KeyMapClass_enum **DRMS_KeyMapClass_t**
- typedef struct DRMS_KeyMap_struct DRMS_KeyMap_t

Enumerations

- enum DRMS_Type_t {

  DRMS_TYPE_CHAR, DRMS_TYPE_SHORT, DRMS_TYPE_INT, DRMS_TYPE_-LONGLONG,

  DRMS_TYPE_FLOAT, DRMS_TYPE_DOUBLE, DRMS_TYPE_TIME, DRMS_TYPE_-STRING,

  DRMS_TYPE_RAW }

  DRMS Data types.

- enum **DRMS_RecLifetime_t** { **DRMS_PERMANENT**, **DRMS_TRANSIENT** }
- enum **DRMS_RecordQueryType_struct** { **kRecordQueryType_DRMS** = 0, **kRecordQueryType_DSDS**, **kRecordQueryType_LOCAL** }
- enum **DRMS_Link_Type_t** { **STATIC_LINK**, **DYNAMIC_LINK** }
- enum DRMS_Segment_Scope_t { DRMS_CONSTANT, DRMS_VARIABLE, DRMS_VARDIM }

  DRMS segment scope types.

- enum **DRMS_KeyMapClass_enum** { **kKEYMAPCLASS_DEFAULT** = 0, **kKEYMAPCLASS_-DSDS** = 1, **kKEYMAPCLASS_LOCAL** = 2, **kKEYMAPCLASS_SSW** = 3 }

Functions

- DRMS_Type_t **drms_str2type** (const char *)
- const char * **drms_type2str** (DRMS_Type_t type)
- int **drms_missing** (DRMS_Type_t type, DRMS_Type_Value_t *val)
- int **drms_copy_db2drms** (DRMS_Type_t drms_type, DRMS_Type_Value_t *drms_dst, DB_-Type_t db_type, char *db_src)
- void **drms_copy_drms2drms** (DRMS_Type_t type, DRMS_Type_Value_t *dst, DRMS_Type_-Value_t *src)

- DB_Type_t **drms2dbtype** (DRMS_Type_t type)
- int **drms_sizeof** (DRMS_Type_t type)
- void * **drms_addr** (DRMS_Type_t type, DRMS_Type_Value_t *val)
- int **drms_strval** (DRMS_Type_t type, DRMS_Type_Value_t *val, char *str)
- int **drms_sprintfval** (char *dst, DRMS_Type_t type, DRMS_Type_Value_t *val, int internal)
- int **drms_sprintfval_format** (char *dst, DRMS_Type_t type, DRMS_Type_Value_t *val, char *format, int internal)
- int **drms_printfval** (DRMS_Type_t type, DRMS_Type_Value_t *val)
- int **drms_sscanf** (char *str, DRMS_Type_t dsttype, DRMS_Type_Value_t *dst)
- int **drms_convert** (DRMS_Type_t dsttype, DRMS_Type_Value_t *dst, DRMS_Type_t srctype, DRMS_Type_Value_t *src)
- int **drms_convert_array** (DRMS_Type_t dsttype, char *dst, DRMS_Type_t srctype, char *src)
- char **drms2char** (DRMS_Type_t type, DRMS_Type_Value_t *value, int *status)
- short **drms2short** (DRMS_Type_t type, DRMS_Type_Value_t *value, int *status)
- int **drms2int** (DRMS_Type_t type, DRMS_Type_Value_t *value, int *status)
- long long **drms2longlong** (DRMS_Type_t type, DRMS_Type_Value_t *value, int *status)
- float **drms2float** (DRMS_Type_t type, DRMS_Type_Value_t *value, int *status)
- double **drms2double** (DRMS_Type_t type, DRMS_Type_Value_t *value, int *status)
- double **drms2time** (DRMS_Type_t type, DRMS_Type_Value_t *value, int *status)
- char * **drms2string** (DRMS_Type_t type, DRMS_Type_Value_t *value, int *status)
- int **drms_printfval_raw** (DRMS_Type_t type, void *val)
- void **drms_byteswap** (DRMS_Type_t type, int n, char *val)
- void **drms_memset** (DRMS_Type_t type, int n, void *array, DRMS_Type_Value_t val)
- int **drms_daxpy** (DRMS_Type_t type, const double alpha, DRMS_Type_Value_t *x, DRMS_Type_Value_t *y)
- int **drms_equal** (DRMS_Type_t type, DRMS_Type_Value_t *x, DRMS_Type_Value_t *y)

Variables

- char * **drms_type_names** [ ]
- char * **drms_segmentscope_names** [ ]

8.11.2 Define Documentation

8.11.2.1 #define DRMS_MAXNAMELEN (32)

Maximum string length of series names, record name, keyword name, link name, etc. !! Series name needs a new define that is larger than 32 chars. !!

Examples:

drms_keymap_ex1.c.

Definition at line 25 of file drms_types.h.

Referenced by drms_insert_records(), drms_keymap_create(), drms_keymap_parsetable(), drms_populate_record(), drms_segment_createinfocon(), drms_template_record(), and drms_template_segments().

8.11.3 Typedef Documentation

8.11.3.1 typedef struct DRMS_KeyMap_struct DRMS_KeyMap_t

DRMS keymap struct reference

Definition at line 811 of file drms_types.h.

8.11.3.2 typedef struct DRMS_Record_struct DRMS_Record_t

DRMS record struct reference

Definition at line 371 of file drms_types.h.

8.11.3.3 typedef struct DRMS_Session_struct DRMS_Session_t

DRMS session struct reference

Definition at line 186 of file drms_types.h.

8.11.4 Enumeration Type Documentation

8.11.4.1 enum DRMS_Segment_Scope_t

DRMS segment scope types.

Enumerator:

DRMS_CONSTANT Indicates data is constant across records.
DRMS_VARIABLE Indicates data dimension structure is constant across records.
DRMS_VARDIM Indicates data dimension structure varies across records.

Definition at line 609 of file drms_types.h.

8.11.4.2 enum DRMS_Type_t

DRMS Data types.

Enumerator:

DRMS_TYPE_CHAR DRMS char type.
DRMS_TYPE_SHORT DRMS short type.
DRMS_TYPE_INT DRMS int type.
DRMS_TYPE_LONGLONG DRMS longlong type.
DRMS_TYPE_FLOAT DRMS float type.
DRMS_TYPE_DOUBLE DRMS double type.
DRMS_TYPE_TIME DRMS time type.
DRMS_TYPE_STRING DRMS string type.
DRMS_TYPE_RAW DRMS raw type.

Definition at line 61 of file drms_types.h.

8.12 base/libs/cmdparams/cmdparams.h File Reference

8.12.1 Detailed Description

Functions to get values in appropriate form from parsed command line and module arguments list.

These functions return values of the appropriate type from the hash list of parameters expected to be available to a module as a global variable.

See also:

module module_args

Definition in file cmdparams.h.

```
#include "hash_table.h"
#include <stdlib.h>
```

Classes

- struct **ModuleArgs_t**
- struct **CmdParams_t_struct**

Defines

- #define **CMDPARAMS_INITBUFSZ** (4096)
- #define **CMDPARAMS_MAXLINE** (1024)
- #define **CMDPARAMS_MAXNEST** (10)
- #define **CMDPARAMS_SUCCESS** (0)
- #define **CMDPARAMS_FAILURE** (-1)
- #define **CMDPARAMS_QUERYMODE** (-2)
- #define **CMDPARAMS_NODEFAULT** (-3)
- #define **CMDPARAMS_UNKNOWN_PARAM** (-4)
- #define **CMDPARAMS_INVALID_CONVERSION** (-5)
- #define **CMDPARAMS_OUTOFMEMORY** (-6)

Typedefs

- typedef struct CmdParams_t_struct **CmdParams_t**

Enumerations

- enum **ModuleArgs_Type_t** {
  ARG_END = 0, **ARG_FLAG**, **ARG_TIME**, **ARG_INT**,
  ARG_FLOAT, **ARG_DOUBLE**, **ARG_STRING**, **ARG_VOID**,
  ARG_INTS, **ARG_FLOATS**, **ARG_DOUBLES**, **ARG_NUME**,
  ARG_DATASET, **ARG_DATASERIES**, **ARG_NEWDATA**, **ARG_NUMARGS** }

Functions

- int cmdparams_parse (CmdParams_t *parms, int argc, char *argv[ ])
- int **cmdparams_parsefile** (CmdParams_t *parms, char *filename, int depth)
- int **cmdparams_exists** (CmdParams_t *parms, char *name)
- void **cmdparams_printall** (CmdParams_t *parms)
- char * **cmdparams_getarg** (CmdParams_t *parms, int num)
- void **cmdparams_usage** (char *name)
- int **cmdparams_numargs** (CmdParams_t *parms)

[cmd]params_isflagset

Check the existence of a flag parameter.

- int cmdparams_isflagset (CmdParams_t *parms, char *name)
- int **params_isflagset** (CmdParams_t *parms, char *name)

cmdparams_get_XXX

Functions to return the value of the parameter name *from the set of cmd-line parameters in* parms.

- int8_t cmdparams_get_int8 (CmdParams_t *parms, char *name, int *status)
- int16_t **cmdparams_get_int16** (CmdParams_t *parms, char *name, int *status)
- int32_t **cmdparams_get_int32** (CmdParams_t *parms, char *name, int *status)
- int64_t **cmdparams_get_int64** (CmdParams_t *parms, char *name, int *status)
- float **cmdparams_get_float** (CmdParams_t *parms, char *name, int *status)
- double **cmdparams_get_double** (CmdParams_t *parms, char *name, int *status)
- int **cmdparams_get_int** (CmdParams_t *parms, char *name, int *status)
- char * cmdparams_get_str (CmdParams_t *parms, char *name, int *status)
- double cmdparams_get_time (CmdParams_t *parms, char *name, int *status)

params_get_XXX

Functions to return the value of the parameter name *from the set of cmd-line parameters in* parms. *These functions are equivalent to the* cmdparams_get_XXX *functions, but without the third argument for a status value.*

- char * params_get_str (CmdParams_t *parms, char *name)
- int8_t **params_get_int8** (CmdParams_t *parms, char *name)
- int16_t **params_get_int16** (CmdParams_t *parms, char *name)
- int32_t **params_get_int32** (CmdParams_t *parms, char *name)
- int64_t **params_get_int64** (CmdParams_t *parms, char *name)
- float **params_get_float** (CmdParams_t *parms, char *name)
- double **params_get_double** (CmdParams_t *parms, char *name)
- double **params_get_time** (CmdParams_t *parms, char *name)
- char **params_get_char** (CmdParams_t *parms, char *name)
- short **params_get_short** (CmdParams_t *parms, char *name)
- int **params_get_int** (CmdParams_t *parms, char *name)

8.12.2 Function Documentation

8.12.2.1 int8_t cmdparams_get_int8 (CmdParams_t * *parms*, char * *name*, int * *status*)

Returns the value in the parameters list associated with the specified name. If there is no entry with that name or if the type does not correspond to *XXX*, then the value representing missing (fill) data for that type is returned, and the *status* value is set to a non-zero code.

On a faliure such as inability to locate the named key or to successfully parse the associated value string according to the prescribed datatype, returns the MISSING value defined for the type.

Definition at line 684 of file cmdparams.c.

8.12.2.2 char* cmdparams_get_str (CmdParams_t * *parms*, char * *name*, int * *status*)

Returns a pointer to the associated string. On a failure, such as inability to locate the named key or to successfully parse the associated value string according to the prescribed datatype, returns a NULL pointer.

Bug

If cmdparams_get_str (which is called by all the other versions of *cmdparams_get_XXX*) cannot find a hash table entry for the requested name, it will use the corresponding environment variable, if it exists.

Definition at line 657 of file cmdparams.c.

8.12.2.3 double cmdparams_get_time (CmdParams_t * *parms*, char * *name*, int * *status*)

Returns a double representing the internal time representation of a parsed date-time string.

Definition at line 857 of file cmdparams.c.

8.12.2.4 int cmdparams_isflagset (CmdParams_t * *parms*, char * *name*)

Returns the value 0 if either the corresponding parameter is 0 or does not exist; otherwise it returns the integer value associated with the parameter (normally -1 if the argument is of type ARG_FLAG). NOTE: These functions are identical.

Definition at line 678 of file cmdparams.c.

8.12.2.5 int cmdparams_parse (CmdParams_t * *parms*, int *argc*, char * *argv*[ ])

Parse command line and module arguments struct and create hashed parms list. This function, ordinarily called by the jsoc_main or other driver program, but accessible from any program, parses the command line (or other array of strings described by *argc* and *argv!*), included files if any, and the global module_args struct to create a hash table of all arguments available in the *parms* struct. Parsing obeys the following rules:

1. All tokens after the first in the command line (*argv*) are parsed for include file references, flags, or parameter name value pairs.

2. Included files, described by tokens of the form in either the command line or an included file, are parsed according to the same rules as the command line.

3. The declared module_args list is scanned, and all named parameters not already assigned values are assigned their declared defaults, if any.

If any declared module argument cannot be assigned a value by the above rules, processing of the argument list ceases and the function returns the value -1. The exception is for arguments declared to be of type ARG_FLAG, which are simply not set.

All values in the parms struct are strings, regardless of the declared type of the argument. For arguments of type ARG_NUME, the value from parsing will be string-compared with the set of valid strings in the argument range, and then set to the character representation of the numeric order of the (first) matching value in the range, counting from 0.

Command-line tokens can take any of the following forms:

–name	(name cannot begin with a digit)
-chars	where chars is a set of one or more characters, all of which are set as key names with the value "1"
name=[]value	white space following the = is optional
@filename	specifies name of file containing tokens to be parsed following same rules as command line
value	assigned to successive unnamed argument $n

If a parameter value (or name) has embedded white space, it must be quoted.

An extern module_args declaration is required, though it can be empty. A member with a module_args.type of ARG_END, must terminate the the list of parsed members; any members following it in the declarator will be ignored.

Bug

Multiple assignments to a given parameter name could result in unpredictable values. The last token in the command line should be parsed last and take precedence. Thus, for example, an assignment following an @filename declaration will superseed the assignment in the corresponding file, and *vice* versa
Parsing of tokens in included files does not properly deal with embedded white space in quoted strings.

Definition at line 307 of file cmdparams.c.

8.12.2.6 char* params_get_str (CmdParams_t * *parms*, char * *name*)

This function is completely analogous to the *cmdparams_get_XXX* version, except that no status is returned.

Definition at line 879 of file cmdparams.c.

8.13 base/libs/dstruct/hcontainer.h File Reference

8.13.1 Detailed Description

Definition in file hcontainer.h.

```
#include "hash_table.h"
#include "jsoc.h"
```

Classes

- struct **HCBuf_struct**
- struct HContainer_struct

  HContainer struct.

- struct **HIterator_struct**

Defines

- #define **HCON_INITSIZE** 2

Typedefs

- typedef struct HCBuf_struct **HCBuf_t**
- typedef struct HContainer_struct HContainer_t

  HContainer struct reference.

- typedef struct HIterator_struct **HIterator_t**

Functions

- void **hcon_init** (HContainer_t *hc, int datasize, int keysize, void(*deep_free)(const void *value), void(*deep_copy)(const void *dst, const void *src))
- void * **hcon_allocslot_lower** (HContainer_t *hc, const char *key)
- void * **hcon_allocslot** (HContainer_t *hc, const char *key)
- void * **hcon_index2slot** (HContainer_t *hc, int index, HCBuf_t **outbuf)
- void * **hcon_lookup_lower** (HContainer_t *hc, const char *key)
- void * **hcon_lookup** (HContainer_t *hc, const char *key)
- void * **hcon_lookup_ext** (HContainer_t *hc, const char *keyin, const char **keyout)
- void * **hcon_lookupindex** (HContainer_t *hc, int index)
- int **hcon_member_lower** (HContainer_t *hc, const char *key)
- int **hcon_member** (HContainer_t *hc, const char *key)
- void **hcon_free** (HContainer_t *hc)
- void **hcon_remove** (HContainer_t *hc, const char *key)
- void **hcon_print** (HContainer_t *hc)
- void **hcon_printf** (FILE *fp, HContainer_t *hc)
- void **hcon_map** (HContainer_t *hc, void(*fmap)(const void *value))
- void **hcon_copy** (HContainer_t *dst, HContainer_t *src)

- void **hcon_stat** (HContainer_t *hc)
- void **hiter_new** (HIterator_t *hit, HContainer_t *hc)
- void **hiter_rewind** (HIterator_t *hit)
- HContainer_t * **hcon_create** (int datasize, int keysize, void(*deep_free)(const void *value), void(*deep_copy)(const void *dst, const void *src), void **valArr, char **nameArr, int valArrSize)

- void **hcon_destroy** (HContainer_t **cont)
- int **hcon_insert** (HContainer_t *hcon, const char *key, const void *value)
- HIterator_t * **hiter_create** (HContainer_t *cont)
- void **hiter_destroy** (HIterator_t **iter)

Chapter 9

JSOC Example Documentation

9.1 drms_keymap_ex1.c

```
DRMS_KeyMap_t *km = drms_keymap_create();
char buf[DRMS_MAXNAMELEN];

if (km)
{
   if (drms_keymap_parsetable(km, text))
   {
      drms_keyword_getintname_ext("CTYPE", NULL, km, drmskeyname, sizeof(buf));
      /* Do something with drmskeyname*/
   }

   drms_keymap_destroy(&km);
}
```

9.2 drms_record_ex1.c

```
int drms_close (DRMS_Env_t *env, int action)
{
   int status;

   if ((status = drms_closeall_records(env,action)))
   {
      fprintf(stderr, "ERROR in drms_close: failed to close records in cache.\n");
   }

   /*  Close connection to database  */
   drms_disconnect(env, 0);
   drms_free_env(env);
   return status;
}
```

9.3 drms_record_ex2.c

```
#include "drms.h"

int status = 0;
DRMS_RecordSet_t *recordSet = NULL;

recordSet = drms_open_records(drms_env, recSetStr, &status);
if (status == DRMS_SUCCESS && recordSet != NULL)
{
   /* Do something with recordSet here. */
}

if (recordSet != NULL)
{
   drms_close_records(recordSet, DRMS_FREE_RECORD);
}
```

9.4 drms_record_ex3.c

```
#include "drms.h"

DRMS_RecordSet_t *recSet = drms_open_records(drms_env, recSetStr, &status);

if (status == 0 && recSet != NULL)
{
   int nRecs = recSet->n;
}

if (recSet != NULL)
{
   drms_close_records(recSet, DRMS_FREE_RECORD);
}


DRMS_RecordSet_t *recSetClone = drms_clone_records(recSet,
                           DRMS_PERMANENT,
                           DRMS_SHARE_SEGMENTS,
                           &status);

if (status == 0 && recSetClone != NULL)
{
   /* Do something with recSetClone here. */
}

if (recSetClone != NULL)
{
   drms_close_records(recSetClone, DRMS_INSERT_RECORD);
}
```

9.5 drms_record_ex4.c

```
#include "drms.h"

int status = 0;
DRMS_RecordSet_t *recSet = drms_open_records(drms_env, recSetStr, &status);
DRMS_RecordSet_t *recSetClone = NULL;

if (status == DRMS_SUCCESS && recSet != NULL)
{
   DRMS_RecordSet_t *recSetClone =
     drms_clone_records(recSet,
            DRMS_PERMANENT,
            DRMS_SHARE_SEGMENTS,
            &status);

   if (status == 0 && recSetClone != NULL)
   {
      /* Do something with recSetClone here. */
   }
}

if (recSet != NULL)
{
   drms_close_records(recSet, DRMS_FREE_RECORD);
}

if (recSetClone != NULL)
{
   drms_close_records(recSetClone, DRMS_INSERT_RECORD);
}
```

9.6 drms_record_ex5.c

```
#include "drms.h"

int status = 0;
DRMS_Record_t *template = drms_template_record(drms_env,
                                               seriesNameStr,
                                               &status);

if (status == DRMS_SUCCESS && template != NULL)
{
     /* Do something with template here. */
     int nPrime = template->seriesinfo->pidx_num;
     ...
}
```

9.7 drms_series_ex1.c

```
//To check for the existence of a series:
int status = DRMS_SUCCESS;
drms_series_exists(drms_env, seriesout, &status);
if (status == DRMS_ERROR_UNKNOWNSERIES))
{
  fprintf(stderr, "Output series %s doesn’t exist.\n", seriesout);
  return 1;
}
```

9.8 drms_series_ex2.c

```
//To access the primary keyword names:
int nPKeys = 0;
char **pkArray = NULL;
int ikey;

pkArray = drms_series_createpkeyarray(drms_env, dsout, &nPKeys, &status);
if (status == DRMS_SUCCESS)
{
  for (iKey = 0; iKey < nPKeys; iKey++)
  {
    /* Do something with pkArray[ikey] */
  }

  drms_series_destroypkeyarray(&pkArray, nPKeys);
}
```

9.9 drms_series_ex3.c

```
// To check a record's compatibility with a series:
int status = DRMS_SUCCESS;
int error = 0;
int compat = 0;
HContainer_t *matchSegNames = NULL;
XASSERT((matchSegNames = (HContainer_t *)malloc(sizeof(HContainer_t))) != NULL);
compat = drms_series_checkrecordcompat(drms_env,
                                       series,
                                       prototype,
                                       matchSegNames,
                                       &status);
if (!compat)
{
  fprintf(stderr,
          "Output series %s is not compatible with output data.\n",
          series);
          error = 1;
}
...
hcon_destroy(&matchSegNames);
```

9.10 drms_series_ex4.c

```
// To check if each member of a set of segments exists in a series:
int status = DRMS_SUCCESS;
int error = 0;
if (!drms_series_checksegcompat(drms_env, series, segs, nSegs, &status))
{
  if (status == DRMS_SUCCESS)
  {
    fprintf(stderror,
            "One or more segments do not match a segment in series %s.\n",
            series);
            error = 1;
  }
}
```

9.11 drms_series_ex5.c

```
// To check if each member of a set of keys exists in a series:
int status = DRMS_SUCCESS;
int error = 0;
if (!drms_series_checkkeycompat(drms_env, series, keys, nKeys, &status))
{
  if (status == DRMS_SUCCESS)
  {
    fprintf(stderror,
            "One or more keywords do not match a keyword in series %s.\n",
            series);
            error = 1;
  }
}
```

Chapter 10

JSOC Page Documentation

10.1 Useful tips for doxygen in C files

- Doxygen reference documentation The Doxygen manual

10.1.1 An introduction to doxygen markup

10.1.1.1 What to document

All declarations for:

1. typedef
2. struct
3. enum
4. functions

This will enable doxygen to link all parameter types to the declarations every time the type is used in a function - very helpful to new developers.

10.1.1.2 Private files

If your declarations are in separate files, like private header files, a simple block can still be linked into doxygen as long as the file is identified to doxygen using a '\file' section:

\file filename.h

\brief one-liner summary of the file purpose

\author the usual copyright statement

10.1.1.3 How to document

Every doxygen comment block starts with an adapted comment marker. You can use an extra slash /// or an extra asterisk. Blocks end in the usual way. Doxygen accepts commands using a backslash.

To put a description with each function or structure, use '\brief' End the brief description with a blank line. The rest of the documentation will then be shown in the body of the doxygen page.

Commands may begin with \ or @

10.1.1.4 Extras

1. Start a line with a hyphen to start a list - the indent determines the nesting of the list:
    - To create a numbered list, use -# e.g. for a sublist:
        (a) start a numbered list
    - revert to previous list

End the list with a blank line. Use :: at the start of a function or structure to link to the page for that function in the doxygen documentation. e.g. qof_class_foreach

Use the param command to describe function parameters in the text.

Use the 'back reference' to document enumerator values:

enum testenum {

enum_one $**<$ less than marker tells doxygen to use this line to document enum_one.

10.1.1.5 Editing Doxygen configuration

To edit the doxygen configuration, you can use:

cd doc

emacs doxygen.cfg &

10.2 Doxygen reference documentation

The Doxygen web site (http://www.stack.nl/~dimitri/doxygen/) has a complete user manual. For the impatient, here are the most interesting sections:

- How to write grouped documentation for files, functions, variables, etc.: http://www.stack.nl/~dimitri/doxygen/grouping.html . Do not forget to add a file documentation block (@file) at the top of your file. Otherwise, all documentation in that file will *not* appear in the html output.

- List of the special commands you can use within your documentation blocks: http://www.stack.nl/~dimitri/doxygen/commands.html

10.3 Bug List

Member cmdparams_get_str If cmdparams_get_str (which is called by all the other versions of *cmdparams_get_XXX*) cannot find a hash table entry for the requested name, it will use the corresponding environment variable, if it exists.

Member cmdparams_parse Multiple assignments to a given parameter name could result in unpredictable values. The last token in the command line should be parsed last and take precedence. Thus, for example, an assignment following an @filename declaration will superseed the assignment in the corresponding file, and *vice* versa

Parsing of tokens in included files does not properly deal with embedded white space in quoted strings.

Group set_keys At present updates of segment files fail. Please use only with the *-c* flag and prime keys when inserting files into generic type data segments.

Group show_keys The program will produce superflous and non-meaningful output if called with the *-p* flag and *seglist* is provided on the command line.

Index