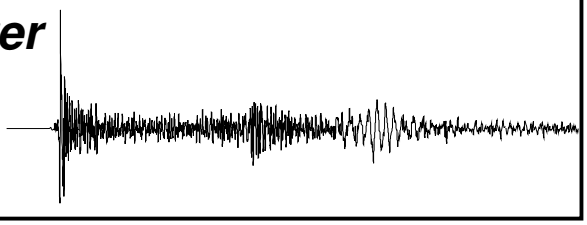
NRDC Information Product Software Manual

Version 1.0
August 1, 1992

Prepared by

IRIS's *Joint Seismic Program Center*

Department of Physics
University of Colorado at Boulder
Campus Box 583
Boulder, CO 80309-0583



Distributed by
IRIS DMC

IRIS DATA MANAGEMENT SYSTEM

Incorporated Research Institutions for Seismology
Data Management Center
1408 NE 45th Street
2nd Floor
Seattle, Washington 98105

Manual Pages

Commands

db2sac, db2ah (1)
: conversion routine for css information

dbcheck (1)
: scans CSS 3.0 table files for corrupted records

dbcp (1)
: copy a CSS database

dblook (1)
: peruse a CSS database

dbpick (1)
: review, pick arrivals, and edit a CSS relational database

dbresponse (1)
: prints response data for selected data traces

e2h (1)
: conversion program for epoch to human times

h2e (1)
: conversion program for human to epoch times

scv2sac, scv2ah (1)
: conversion routines for scv information

C library routines

banner (3)
: print a standard banner

complain, die, register_error, clear_register (3)
: handle error messages in a standard way

db30_create (3)
: create and read a CSS 3.0 database object

DBL_add_tuple, DBL_get_tuple, DBL_put_tuple (3)
: manipulate relation tuples

DBL_compose_filename (3)
: generate the path for an external file referenced in a table

DBL_free (3)
: free all resources associated with a database

DBL_get_attrs (3)
: get one or more attributes from a single database tuple

DBL_get_nrels, DBL_get_ntuples, DBL_exist_db, DBL_exist_rel, DBL_get_relid (3)
: database utility functions

DBL_nextid (3)
: return a unique id for a field in a CSS table

DBL_put_attrs (3)
: put one or more attributes into a single database tuple

doy2mday, mday2doy, h2e, e2h (3)
: conversion of time between month/day, day of year, and epoch time

find_path, find_file_path (3)
search for CSS table or file along JSPC_DBPATH

get_argument, get_directory, get_input_file, get_output_file, last_input_file, last_output_file, next_file, readable_file, writable_file (3)
Standard command line parsing routines

get_wfdisc30_trace_data (3)
read trace data for CSS 3.0 wfdisc table

intro (3)
Station

introcss30 (3)
Utilities for manipulating CSS v. 3.0 flat file databases

read_response, write_response, eval_response, free_response (3)
manipulate response files

SCV_add_arrival (3)
add a new arrival to the station

SCV_create (3)
create one or more Station

SCV_free, SCV_free_all (3)
free memory associated with SCV's

SCV_get (3)
get one or more Station

SCV_get_arrival, ARR_get (3)
get attributes from a Station

SCV_get_segment (3)
get attributes from a Station

SCV_get_trace, SCV_free_trace (3)
manipulate data samples for a Station

SCV_put_arrival, ARR_put (3)
set attributes for a Station

SCV_read_wfs (3)
read waveform sample data into memory

SCV_sift_stachan (3)
sift a set of station

time_string2epoch (3)
conversion to epoch time from an ascii string

FORTRAN library routines

db30create (3f)
create one or more Station

dbaddtuple (3f)
add a NULL tuple to a relation, FORTRAN version

dbgetattrs (3f)
get one or more attributes from a single database tuple, FORTRAN version

dbputattrs (3f)
put one or more attributes into a single database tuple, FORTRAN version

doy2mday, mday2doy, h2e, e2h (3f)
: conversion of time between month/day, day of year, and epoch time

getwf (3f)
: read trace data for CSS 3.0 wfdisc table

putwf (3f)
: write trace data for CSS 3.0 wfdisc table

scvcreate (3f)
: create one or more Station

scvget (3f)
: get one or more Station

scvgetarrival (3f)
: get Station

scvreadwfs, scvgettrace, scvfreewfs (3f)
: read or free all waveforms associated with an SCV handle

scvselect (3f)
: select scvs satisfying sift keys

File formats

response (5)
: format of instrument response files

NAME

db2sac, db2ah – conversion routine for css information

SYNOPSIS

db2sac [-e] [-t *type***] [-s** *format***] -i** *input_db* **-o** *output_db* **-w** *wf_dir*

db2ah [-e] [-t *type***] [-s** *format***] -i** *input_db* **-o** *output_db* **-w** *wf_dir*

DESCRIPTION

Db2sac and **db2ah** are conversion utilities that convert *CSS* format files into *SAC* or *AH* format, and creates a new *CSS* database incorporating these files. Note that the *CSS* database remains consistant, and programs using a *CSS* database should continue to run. The *CSS* files can be in either *css-2.8* or *css-3.0* format. More data formats will be supported in the future.

Each *CSS* database can create a number of *SAC*/*AH* files. These files are named using the following convention:

YYYYJJJHHMMSS.SS.STN.CHAN

Where the fields have the following meanings:

YYYY	Year
JJJ	Julian Day
HH	Hour
MM	Minute
SS.SSSS	Seconds.Fractional-Seconds
STN	Station
CHAN	Channel

OPTIONS

-e | **-env** Copies database files reached through the environment variable *JSPC_DBPATH*. The default is to only copy the files specified by the **-i** *input_db* option.

-t [*rsavd***]** | **-type [***rsavd***]**
Specifies the *data_type* of the waveform data supporting the Station-Channel View. The type is specified by listing the first character of each of the desired options, the order is not important. The type options supported are *acceleration*, *velocity*, and *displacement*, for either *real* or *synthetic* data. The default is '**rv**', for *real* data measured in units of *velocity*.

-s *schema* | **-schema** *schema*
Specifies the *schema* (format) for the data supporting the Station-Channel View. Currently the only *schema* supported are *''css-2.8''* and *''css-3.0''* The default is *''css-3.0''*

-o *file* | **-output** *file*
Specifies the output *file* name. If no -o option is present, the output is written to the current directory.

-i *file* | **-input** *file*
Specifies the input *file* name. If no -i option is present, the first non-option argument is used instead.

-h | **-help**
Gives a summary of the command usage.

-v | **-version**
Print the version number and last modification date.

EXAMPLE

Here is a simple example that writes two *SAC* files from input css-3.0 files. The data is *synthetic*, in units of *displacement*.

```
shell% ls
1987159/                1987159.arrival         1987159.wfdisc

shell% db2sac -t sd -s css-3.0 -i 1987159 -o foo -w .
DBL_read: Reading.done
DBL_read: Reading.done
DBL_read: Reading.done
Translating...
Writing XXX, 06, 220750 samples (BINARY),
starting 1987 159 13:32:05:5000 UT
Translating...
Writing XXX, 04, 220750 samples (BINARY),
starting 1987 159 13:32:05:5000 UT

shell% ls
1987159133205.50.XXX.04         1987159133205.50.XXX.06
1987159/                        1987159.wfdisc
1987159.arrival                 foo.arrival
foo.wfdisc
```

ENVIRONMENT

JSPC_DBPATH=*dbpath*

Defines the sequence of locations to search when looking for database files. See **find_path(3)** for more details.

SEE ALSO

scv2sac(1), find_path(3)

BUGS

The file names are **gross**, but will stay within the 32 character limit required to be compatible with *CSS-3.0*.

NAME

dbcheck – scans CSS 3.0 table files for corrupted records

SYNOPSIS

dbcheck *database_name*

DESCRIPTION

Each table in the database specified by the *database_name* argument and JSPC_DBPATH is inspected for records which have a bad length. Such records are often introduced inadvertently by hand editing of the files. Records which do not have the correct length are printed.

EXAMPLE

```
tagg% dbcheck nrdc
checking /spc/slick3/NRDC/data/tables/nrdc.affiliation ..
record#3 is bad:
NRDC      KSU    05/08/92
checking /spc/slick3/NRDC/data/tables/nrdc.assoc ..
checking /spc/slick3/NRDC/data/tables/nrdc.instrument ..
checking /spc/slick3/NRDC/data/tables/nrdc.network ..
checking /spc/slick3/NRDC/data/tables/nrdc.origin ..
checking /spc/slick3/NRDC/data/tables/nrdc.sensor ..
checking /spc/slick3/NRDC/data/tables/nrdc.site ..
checking /spc/slick3/NRDC/data/tables/nrdc.sitechan ..
checking /spc/slick3/NRDC/data/tables/nrdc.sregion ..
tagg%
```

SEE ALSO

find_path(3), introcss30(3)

BUGS AND CAVEATS

The confusion that results with JSPC_DBPATH when databases do not have unique names applies here. For example, if a database has two files with identical names, one in the current directory and one in a directory in JSPC_DBPATH, only the file in the current directory is checked.

AUTHOR

Tom Montague

NAME

dbcp – copy a CSS database

SYNOPSIS

dbcp [-e] [-s *schema***] -i** *input_db* **-o** *output_db* **-w** *wave_dir*

DESCRIPTION

Dbcp copies a CSS (Center for Seismic Studies) relational database, and the associated wave-form files, to a new location. Both version 2.8 and 3.0 databases can be copied with this program.

It is recommended that **dbcp** be used instead of **cp** since the CSS schema contains directory names, and may be corrupted by a simple file copy.

OPTIONS

-e Copies database files reached through the environment variable *JSPC_DBPATH*. The default is to only copy the files specified by the **-i** *input_db* option.

-s *schema*

Specifies the CSS schema describing the database to copy. Currently *css-2.8* and *css-3.0* are supported. The default is *css-3.0*.

-i *input_db*

The name of the database to copy. The database name is defined as the name of the flat relation files without the .relname suffix. For example, if the .wfdisc relation file name is ./foo.wfdisc then ./foo is the database name.

-o *output_db*

The name of the database to write. The database name is defined as the name of the flat relation files without the .relname suffix. For example, if the .wfdisc relation file name is ./foo.wfdisc then ./foo is the database name.

-w *wave_dir*

The location for the waveform files. If *wave_dir* is given as a relative path, it is assumed to start from the *output_db* directory. It is best, if possible, to use a relative pathname.

ENVIRONMENT

JSPC_DBPATH=*dbpath*

Defines the sequence of locations to search when looking for database files. See **find_path(3)** for more details.

SEE ALSO

dblook(1), dbpick(1), css2sac(1), find_path(3)

NAME

dblook – peruse a CSS database

SYNOPSIS

dblook [-28] [-raw_time] *dbname*

DESCRIPTION

Dblook is an X-windows based interactive graphical program for viewing CSS (Center for Seismic Studies) version 2.8 and 3.0 relational databases.

OPTIONS

dbname The name of the subject database. The database name is defined as the name of one of the flat relation files without the .relname suffix. For example, if the .wfdisc relation file name is ./foo.wfdisc then ./foo is the database name. This argument is required.

-28 The input file is in the CSS-2.8 format. If this flag is missing **dblook** will first try to open the database as CSS-3.0 format.

-raw_time

Display times as CSS-style epoch times instead of the default which is to display times in human readable from.

PROGRAM INTERACTION

All program interactions are produced by pressing the following buttons with the mouse.

Quit	Exit from the program. All windows are closed.
suffix-name(*rows*)	Create a window displaying the 'suffix-name' table. **Suffix-name** is the file extension identifying the associated flat file.
Arrange	Display a check-off list of the information contained in the table. The displaye is controlled by checking/unchecking the list entries.
Done	Close this window.
Apply	Close this window, applying the changes selected to the display.
Cancel	Close this window, without having any effect on the previous configuration.
Default	Select the default display. The default is to display all information.
Clear	Remove all selected items. This is the opposite of the **Default** button.

Additionally, pressing the **Left Mouse** button on a row of the *wfdisc* table generates a plot of the associated waveform if possible.

ENVIRONMENT

JSPC_DBPATH=*dbpath*

Defines the sequence of locations to search when looking for database files. See **find_path(3)** for more details.

SEE ALSO

dbpick(1), find_path(3)

NAME

dbpick – review, pick arrivals, and edit a CSS relational database

SYNOPSIS

dbpick [–sc *sta:chan***] [–ts** *tstart***] [–te** *tend***] [–tw** *twin***] [–exist]** *dbname*

DESCRIPTION

dbpick is an X-windows based interactive graphical program for creating and/or editing phase arrivals in a CSS (Center for Seismic Studies) relational database. The database must be represented as a set of flat files, each file corresponding to one of the CSS relations. Currently the program will work with versions 2.8 and 3.0 of the database structure. The flat files may be either in normal ASCII form or they can be in compiled form (a binary image of the internal C tuple structures). Changes to the subject database occur in a transactional manner, i.e. editing causes immediate changes to the database flat files without copying to new files. The user should be careful to back up a database, by using **dbcp** for example, prior to using **dbpick**.

COMMAND LINE ARGUMENTS

dbname The name of the subject database. The database name is defined as the name of one of the flat relation files without the .relname suffix. For example, if the .wfdisc relation file name is ./foo.wfdisc then ./foo is the database name. This argument is required.

–sc *sta:chan*

A station-channel sift key. This argument specifies the subset of station-channel data streams that will be available for analysis. This argument is optional and if it is not specified, then all station-channel values will be used. The asterisk (∗) character can be used in place of either *sta* or *chan* which matches anything. Thus -sc BAY:∗ would mean all channels for station BAY or -sc ∗:sz would mean all stations for channel sz. In addition, normal UNIX regular expression matching is used to match strings (ala ed or ex). The user should take care to deal properly with the shell interpreter when using the various metacharacters, as they can be substituted before program execution. If using the C-shell, a safe way to handle this is to place *sta:chan* between single quotes as follows: **–sc** *'sta:chan'*

–ts *tstart* A data sift start time. This argument specifies the subset of arrivals and waveform segments that will be available for analysis. Only arrivals and waveform segments with time later than *tstart* will be used. This argument is optional and if it is not specified, then waveform segment and arrival times will not be subject to start time sifting. The start time, *tstart*, can be specified in several different ways:

1. An epoch time in seconds since 00:00:00 1 Jan 1970 GMT.
2. A character string in 'year:month:day:hour:minute:second' form.
3. A character string in 'julianday:hour:minute:second' form.

–te *tend* A data sift end time. This argument specifies the subset of arrivals and waveform segments that will be available for analysis. Only arrivals and waveform segments with time earlier than *tend* will be used. This argument is optional and if it is not specified, then waveform segment and arrival times will not be subject to start time sifting. The end time, *tend*, can be specified in several different ways:

1. An epoch time in seconds since 00:00:00 1 Jan 1970 GMT.
2. A character string in 'year:month:day:hour:minute:second' form.
3. A character string in 'julianday:hour:minute:second' form.

–tw *twin* An initial time window in seconds for displaying the data. This argument is optional and, if not specified, the entire time window of the data, after sifting, is used.

–noexist A waveform existence flag. If this flag is given in the command line, then all station-channel streams will be displayed even those for which there are no existing waveform .w files.

RESOURCE DATABASE

This program makes use of the X11 resource database software and a number of program parameters can be set by the user through the use of X11 resource files. The user can specify these resource parameters either through the normal X-window resource path (in *.Xresource or .Xdefaults* file or any other file that is loaded with **xrdb**), or in special resource files with name *.dbpickrc* that are always loaded by **dbpick** at run time. The program first looks for *.dbpickrc* in the user's home directory and then in the current working directory. If *.dbpickrc* files reside in both directories, then the contents of both files are read with the file in the current working directory overwriting parameters from the file in the user's home directory. A typical *.dbpickrc* file would be as follows:

```
dbpick.traceWindow.twin: 300.0
dbpick.filter1.label:        0.1 HP
dbpick.filter1.tpad:         100.0
dbpick.filter1.lcoFreq:      0.1
dbpick.filter1.lcoOrder:  3
dbpick.filter1.ucoFreq:      0.0
dbpick.filter1.ucoOrder:  0
```

The resource names and values are defined as follows.

Resource Name	**Value**
dbpick.foreground	The trace display foreground color (on color terminals only). This can be specified either in the normal X-window fashion (e.g. 'red', '#ff348a'), or as a hue-lightness-saturation floating point triad (e.g. '240.0 0.5 1.0' for pure blue).
dbpick.background	The trace display background color (on color terminals only). This is specified as with the trace display foreground color.
dbpick.arrival.foreground	The normal arrival flag foreground color (on color terminals only). This is specified as with the trace display foreground color.
dbpick.arrival.background	The normal arrival flag background color (on color terminals only). This is specified as with the trace display foreground color.
dbpick.selArrival.foreground	The 'selected' arrival flag foreground color (on color terminals only). This is specified as with the trace display foreground color.
dbpick.selArrival.background	The 'selected' arrival flag background color (on color terminals only). This is specified as with the trace display foreground color.
dbpick.traceWindow.twin	The initial trace display time window in seconds.
dbpick.filter#.label	The pull-down menu label for the #th filter. The '#' character is an integer in the range of 1 to 11 and represents the filter number.
dbpick.filter#.tpad	The filter transient time in seconds for the #th filter. This is used to lead the requested time window so that the filter transient has had a chance to decay.
dbpick.filter#.lcoFreq	The filter lower cutoff frequency in Hertz for the #th filter. If this is 0.0, then the filter is low-pass.
dbpick.filter#.lcoOrder	The filter lower cutoff order for the #th filter. This is the number of poles used in the low-band stage of the Butterworth filter.
dbpick.filter#.ucoFreq	The filter upper cutoff frequency in Hertz for the #th filter. If this is 0.0, then the filter is high-pass.
dbpick.filter#.ucoOrder	The filter upper cutoff order for the #th filter. This is the number of poles used in the high-band stage of the Butterworth filter.

PROGRAM INTERACTION

Upon initial startup the station-channel values after sifting are displayed, along with the total number of existing waveform data samples, and the user is queried about continuing. If the user answers 'n', then the program exits and the user can try different sifting keys. If the user answers 'y', then the user may be asked if the trace samples should be read into core memory. If there are less than a total of 500,000 data samples, then the waveform data is automatically read into core memory, otherwise the user is asked if the waveform data should be read into memory. Reading the waveform data into memory will speed up the program, assuming there is enough available memory to handle the data. After these initial questions the data is displayed in a graphical window and the program goes into command input mode. Command input mode is indicated by the prompt *dbpick>* displayed in the original text window and whenever this prompt is displayed the user may either enter text commands at the prompt or interact directly with the various graphical display windows through the mouse and keyboard. The currently implemented text commands are as follows:

fw Time scroll the primary display to the first existing waveform.

nw Time scroll the primary display to the next existing waveform.

pw Time scroll the primary display to the previous existing waveform.

fa Time scroll the primary and secondary displays to the first phase arrival with an existing waveform.

na Time scroll the primary and secondary displays to the next phase arrival with an existing waveform.

pa Time scroll the primary and secondary displays to the previous phase arrival with an existing waveform.

tfit Toggle the time window fit mode. When time window fit mode is enabled, the trace display time window is automatically adjusted so that it exactly fits the waveform segments being currently displayed.

sfit Fit the trace display in the vertical station-channel direction. This causes all station-channels to be displayed.

sc *sta:chan* Display only those traces that match *sta:chan*. This is the keyboard equivalent of the **-sc** command option, except that all station-channel data streams remain available.

ts *time* Display the station-channel data streams starting at *time*. Time must be in one of the formats specified for the **-ts** command option.

tw *time* Change the primary display time window to *time* seconds.

parr ????Toggle phase arrange mode????

deglitch ????Toggle the deglitcher????

help Display a list of available text commands.

quit Exit the program.

In addition to entering text commands at the command prompt, the user can also enter certain commands directly by typing single keys while the mouse pointer is in the graphical display window. These keys can be thought of as accelerators and for the most part they simply replace typing at the command prompt. The accelerator keys are as follows.

F Time scroll the primary display to the first existing waveform. Same as 'fw' prompt command.

N Time scroll the primary display to the next existing waveform. Same as 'nw' prompt command.

P Time scroll the primary display to the previous existing waveform. Same as 'pw' prompt command.

f Time scroll the primary and secondary displays to the first phase arrival with an existing waveform. Same as ‘fa’ prompt command.

n Time scroll the primary and secondary displays to the next phase arrival with an existing waveform. Same as ‘na’ prompt command.

p Time scroll the primary and secondary displays to the previous phase arrival with an existing waveform. Same as ‘pa’ prompt command.

t Toggle the time window fit mode. Same as the ‘tfit’ prompt command.

s Fit the trace display in the vertical station-channel direction. Same as the ‘sfit’ prompt command.

r Repaint window.

R Redraw and repaint window.

All other commands and interaction take place directly within the graphical display windows. The main display window initially shows all of the selected station-channel data streams with phase arrival flags. Control buttons appear at the top of the window. The effects of mouse pointer and button events in the display windows depend on where the mouse is currently located.

The **Main Window** control buttons are as follows.

Traces Causes the trace display pull-down menu to appear. Items selected from this menu control how the traces are displayed on the window. See **Station Region** for information on selecting traces.

- **Select** Changes the display to contain the selected traces. If no traces were selected then the display will not change.
- **Delete** Changes the display to contain the unselected traces. If no traces were selected then the display will not change.
- **Original** Causes all traces to be displayed.
- **SelectAll** Selects every trace in the display.
- **DeleteAll** Unselects every trace in the display.
- **Zoom** Changes the display to show only the range of selected traces. The other traces are still part of the display, and may be reached with the scroll bar.
- **NewWin** Creates a new display window containing the selected traces.

Amp Causes the amplitude scale mode pull-down menu to appear. Items selected from this menu control how trace amplitudes are scaled within the displayed.

- **Fixed** Displays the traces with fixed amplification.
- **Auto*** ???? Some sort of automatic mumble ????
- **AGC** Applies Automatic Gain Control to the displayed traces.
- **ClipOff** ???? Toggles clipping mode. ????
- **Invert** Reverses the polarity of all of the traces.
- **Gain=1** Displays the traces at normal amplitude
- **Gain x N** Increases the current gain of the traces by N. Note that this is cumulative.
- **Gain / N** Decreases the current gain of the traces by N. Note that this is cumulative.

Filter Causes the filter pull-down menu to be displayed. Items selected from this menu control which, if any, filter is applied to the data before it is displayed. The filter

parameters can be defined through the X11 resource files described previously.

None Displays the traces without any filtering.

Freq **LP** Applies a low-pass filter with the corner frequency at *Freq*.

Range **LP** Applies a band-pass filter with the corner frequencies at *Range*.

Add Arrivals Causes the mouse to be placed in add-arrival mode. While in this mode the mouse buttons have the following effect:

Left Mouse One arrival is added, at the position of the mouse, and the mouse reverts to the normal state.

Middle Mouse The middle button causes the mouse to revert to the normal state, without adding an arrival.

Right Mouse The right button adds an arrival, at the position of the mouse, with the mouse remaining in add-arrival mode.

Done Closes the window containing the button.

The **Trace Region** is the area, excluding the arrival flags, where the traces are displayed Within this region the mouse acts as follows:

Left Mouse The left mouse button causes the traces to be scrolled to the left so that the mouse position is shifted to the left edge of the screen.

Middle Mouse Dragging the middle mouse button causes the traces to follow the movement of the mouse.

Right Mouse The right mouse button causes the traces to be scrolled to the right so that the mouse position is shifted to the right edge of the screen.

The **Station Region** is the area to the left of the traces where the *Station*/*Channel* information is displayed. Within this region the mouse acts as follows:

Left Mouse Pressing the left mouse button in this region causes the specified traces to be selected/unselected.

The **Arrival Region** is the area within an arrival flag. Within this region the mouse acts as follows:

Left Mouse Pressing and ‘‘dragging’’ the left mouse button in this region causes the arrival time to be changed until the button is released.

Middle Mouse Pressing the middle button causes a phase pop-up menu to appear and the selection of a menu item will cause the arrival phase to be changed.

Right Mouse Pressing the right mouse button causes an arrival editing pop-up menu to appear with the following options:

Window Create a new window with the same time scale

Magnify Create a new time magnified window

Range Set the arrival uncertainty

Delete Delete the arrival.

The magnification window allows the user to make more sensitive interactive measurements than would be possible in the original window. All changes in the arrival flags are synchronized among the windows. The secondary windows behave in all aspects like the original window.

The **Scroll Region** is the area around the scroll bars. Within this region the mouse has the following effects:

Right Mouse Pressing the right mouse button within this region causes the scroll pull-down menu to appear. The items and their effects are defined as follows:

Time Fit Rescales the time axis so the entire trace is visible within the

window.

Time ZoomIn Time zoomin. Increases the magnification along the time axis.

Time ZoomOut Time zoomout. Decreases the magnification along the time axis.

Trace Fit Rescales the trace axis so that all traces are visible within the window.

Trace ZoomIn Trace zoomin. Increases the magnification along the trace axis.

Trace ZoomOut Trace zoomout. Decreases the magnification along the trace axis.

ENVIRONMENT

JSPC_DBPATH=*dbpath*

Defines the sequence of locations to search when looking for database files. See **find_path(3)** for more details.

BUGS

Currently the ‘‘Range’’ menu option is disabled, and has no effect on the database.

SEE ALSO

dblook(1), dbcp(1), css2sac(1), find_path(3)

NAME

dbresponse – prints response data for selected data traces

SYNOPSIS

dbresponse [*-l*] *database_name*

DESCRIPTION

Dbresponse selects traces for which to show response data using a station channel sift key (see **SCV_create(3)**). If the -l option is specified, the response file for the selected channels are printed. Otherwise, the program prompts for the frequency range and number of data points for which to show the response.

The *database_name* is the base name of the database without any of the flat-file extensions. If *database_name* was ‘‘foo’’ then the database files would be named ‘‘foo.sitechan’’, ‘‘foo.wfdisc’’, etc.

Responses for each selected data trace are printed separately, along with some associated data. The response table shows the magnitude and phase for the specified frequency range.

EXAMPLE

overend% **dbresponse nrdc/data/proc/1987078/1987078**
Enter Station-Channel sift key: KKL:HSSZ
DBL_read: Reading.done
starting, ending frequency? .01 100
number of points? 5

Database Name (YearDay) : nrdc/data/proc/1987078/1987078
Station Channel Sift Key: KKL:HSSZ

Station Channel (Lat., Long.) Elevation
KKL HSSZ (49.33, 75.38) 1.0
Tstart : 3/19/1987 (day 78) 14:37:56.50
Tend : 3/19/1987 (day 78) 14:38:44.00

(HAng, VAng) Depth Calib Calper
(0.0, 0.0) 0.000e+00 1.592e-02 1.000e-01
Response file is /spc/slick3/NRDC/data/tables/nrdc.resp/GS13.23.rsp

frequency	magnitude	phase (degrees)
0.010000	0.000000	77.54841
0.100000	0.000078	-10.35732
1.000000	0.084160	170.83508
10.000000	1.000000	69.85234
100.000000	2.535741	108.99133

SEE ALSO

SCV_create(3), introscv(3)

BUGS

The starting frequency must be less than the ending frequency, and there must be at least two points specified.

AUTHOR

Tom Montague

NAME

e2h – conversion program for epoch to human times

SYNOPSIS

e2h [-ymndjHMS] [-o *file***]** *epoch* **e2h [-ymndjHMS] [-i** *file***] [-o** *file***]**

DESCRIPTION

E2h converts *epoch* times to *human* times. The *epoch* time is the number of seconds since 00:00 hrs, January 1, 1970. The format for the human time is specified by the command switches. The default is rather verbose.

OPTIONS

-i *file* | **-input** *file*
The name of the input *file* containing the epoch time. If no *file* is specified, then the epoch time will be expected on the command line.

-o *file* | **-output** *file*
The name of the output *file* for the human time. If no *file* is specified, then the results will be written to stdout.

-v | **-version**
Print the version number, and the last modification date.

-h | **-help**
Print a summary of the command usage.

The format for the output is specified by the option switches, and their order. The output options are as follows:

-y | **-year**
Print the *year* as a number.

-m | **-month**
Print the *month* as a number 1-12.

-n | **-name**
Print the *month* as a three letter name.

-d | **-day**
Print the *day* as a number 1-31.

-j | **-julian**
Print the julian *day* as a number 1-366.

-H | **-hour**
Print the *hour* in military time.

-M | **-min**
Print the *minute* as a number 1-59.

-S | **-sec** Print the *second* as a floating point number 0.0-59.999999

EXAMPLE

The following examples all take the epoch time from the command line, and output to stdout. The output would be the same if one, or both, came from a file instead.

```
shell% e2h 812345678
1995 9 Sep 29 272 3 34 38.000000

shell% e2h -year -month -day 812345678
1995 9 29

shell% e2h -dny 812345678
29 Sep 1995
```

```
shell% e2h -HMS 812345678
3 34 38.000000
```

SEE ALSO

h2e(1)

BUGS

The output switches are a hack of sorts...

NAME

h2e – conversion program for human to epoch times

SYNOPSIS

h2e [-y *year***] [-m** *month***] [-d** *day***] [-H** *hour***] [-M** *min***] [-S** *sec***] [-o** *file***]**

DESCRIPTION

H2e converts *human* times to *epoch* times . The *epoch* time is the number of seconds since 00:00 hrs, January 1, 1970. The format for the human time is specified by the command switches.

OPTIONS

-o *file* | **-output** *file*
The name of the output *file* for the epoch time. If no *file* is specified, then the results will be written to stdout.

-v | **-version**
Print the version number, and the last modification date.

-h | **-help**
Print a summary of the command usage.

The format for the input is specified by the option switches. The input options are as follows:

-y *year* | **-year** *year*
The *year* as a number.

-d *day* | **-day** *day*
The *day* as a number 1-31.

-H *hour* | **-hour** *hour*
The *hour* in military time.

-M *minute* | **-min** *minute*
The *minute* as a number 1-59.

-S *second* | **-sec** *second*
The *second* as a floating point number 0.0-59.999999

EXAMPLE

The following examples all output to stdout. The output would be the same if it went to a file instead.

```
shell% h2e -year 2001 -month 12 -day 31 -hour 23 -min 59 -sec 59.999999
1009843200.000000

shell% h2e -y 2001 -m 12 -d 31 -H 23 -M 59 -S 59.0
1009843199.000000

shell% h2e -S 59.0 -M 59 -H 23 -d 31 -m 12 -y 2001
1009843199.000000

shell% h2e -y 2000
946684800.000000
```

SEE ALSO

e2h(1)

BUGS

The input switches are a hack of sorts...

NAME

scv2sac, scv2ah – conversion routines for scv information

SYNOPSIS

scv2sac [-s *format***] [-o** *output-file***] [-i]** *input-file* **scv2ah [-s** *format***] [-o** *output-file***] [-i]** *input-file*

DESCRIPTION

Scv2sac and **scv2ah** are conversion utilities that convert Station-Channel View information into *sac* or *ah* format, respectively. The Station-Channel View informatin can come from *css-2.8* or *css-3.0* data files. More data formats will be supported in the future.

Each Station-Channel View can create a number of *sac* / *ah* files. These files are named using the following convention:

YYYYJJJHHMMSS.SS.STN.CHAN

Where the fields have the following meanings:

YYYY	Year
JJJ	Julian Day
HH	Hour
MM	Minute
SS.SSSS	Seconds.Fractional-Seconds
STN	Station
CHAN	Channel

The *ah* files, written by **scv2ah** are written in an architecture dependent format. They can be converted into the architecture independent *xdr* format by using the **ah2xdr** filter supplied with the *ah* distribution.

OPTIONS

-s *schema* | **-schema** *schema*
Specifies the *schema* (format) for the data supporting the Station-Channel View. Currently the only *schema* supported are *css-2.8* and *css-3.0* The default is *css-3.0*

-o *file* | **-output** *file*
Specifies the output *file* name. If no -o option is present, the output is written to the current directory.

-i *file* | **-input** *file*
Specifies the input *file* name. If no -i option is present, the first non-option argument is used instead.

-h | **-help**
Gives a summary of the command usage.

-v | **-version**
Print the version number and last modification date.

EXAMPLE

Here is a simple example that writes two sac files from input css-3.0 files.

```
shell% ls
1987159/                1987159.arrival         1987159.wfdisc

shell% scv2sac -s css-3.0 /data/1987159
DBL_read: Reading.done
Translating...
Writing XXX, 06, 220750 samples (BINARY),
starting 1987 159 13:32:05:5000 UT
Translating...
Writing XXX, 04, 220750 samples (BINARY),
starting 1987 159 13:32:05:5000 UT
```

```
shell% ls
1987159133205.50.XXX.04          1987159133205.50.XXX.06
1987159/                         1987159.wfdisc
1987159.arrival
```

ENVIRONMENT

JSPC_DBPATH=*dbpath*

Defines the sequence of locations to search when looking for database files. See **find_path(3)** for more details.

SEE ALSO

ah2xdr(1), rdseed(1), find_path(3)

BUGS

The file names are **gross**, but will stay within the 32 character limit required to be compatable with *CSS-3.0*.

NAME

banner – print a standard banner

SYNOPSIS

void banner(*program***,** *version***)**
char ∗*program***;**
char ∗*version***;**

DESCRIPTION

This routine is used to print a header announcing the name of the *program*, the *version*, and the *JSPC* address. It is used by interactive programs to standardize the greeting message.

SEE ALSO

style(5)

NAME

complain, die, register_error, clear_register – handle error messages in a standard way

SYNOPSIS

#include ''stock.h''

void complain(*sysflag***,** *format* **[,** *arg***...])**
int *sysflag***;**
char ∗*format***;**

void die(*sysflag***,** *format* **[,** *arg***...])**
int *sysflag***;**
char ∗*format***;**

void register_error(*sysflag***,** *format* **[,** *arg***...])**
int *sysflag***;**
char ∗*format***;**

void clear_register(*printflag***)**
int *printflag***;**

extern char ∗*Program_Name***;**

DESCRIPTION

This family of routines is intended to standardize the handling of errors, by funneling error messages through a central routine. It provides convenient and concise facilities for announcing problems for both library routines and programs.

Except for **clear_register()**, all these routines take standard printf arguments. All use the global variable *Program_Name*, which should be initialized to contain the (null-terminated) name of the program.

Complain() prints a message immediately, and always returns. If *sysflag* is nonzero, the system error message is also printed.

Die() prints a message immediately, and exits with error code *1*; it never returns. If *sysflag* is nonzero, the system error message is also printed.

Library routines should almost never exit, but should instead return an error code. However, an error code alone may be quite limiting. **Register_error()** allows a library routine to post an error message, which the calling program can choose (or choose not) to print out. Just like **complain()**, it takes a flag to indicate whether to include the system error message, and a standard set of printf arguments. **Clear_register()** clears the existing register of error messages, printing the messages if printflag is nonzero.

FILES

The error messages are printed on stderr.

EXAMPLE

```
if ( (file = fopen ( filename, ''r'' ) ) == NULL )
   complain ( 1, ''Can't open file %s\n'', filename ) ;
```

DIAGNOSTICS

** Too many error messages -- Error Buffer Overflow.

register_error() allows a limited number (20) of error messages to be posted at a time; if more errors are posted, then the earliest are lost, and this message is printed when the error log is printed.

SEE ALSO

perror(1)

AUTHOR

Daniel Quinlan

NAME

db30_create – create and read a CSS 3.0 database object

SYNOPSIS

#include ''DBL_defs.h''

DBLink ∗**db30_create**(*dbname*)
char ∗*dbname***;**

DESCRIPTION

The proceedure **db30_create()** will create a CSS v. 3.0 database object, map it to the external flat file representation and load it with data from any existing external flat files.

ARGUMENTS

dbname (i) The external flat file database name. This is the same as one of the relation file names without the *.relname* suffix.

RETURN VALUE

If **db30_create()** fails, then *NULL* will be returned and an error message is (usually) printed on standard error output. If **db30_create()** succeeds, then a pointer to a database object (**DBLINK** ∗) is returned.

SEE ALSO

introcss30(3)

NAME

DBL_add_tuple, DBL_get_tuple, DBL_put_tuple – manipulate relation tuples

SYNOPSIS

#include ''DBL_defs.h''
#include ''css30_defs.h''

int DBL_add_tuple(*db*, *relid*)
DBLink **db*;
int *relid*;

Tuple *DBL_get_tuple(*db*, *relid*, *ituple*)
DBLink **db*;
int *relid*;
int *ituple*;

int DBL_put_tuple(*db*, *relid*, *tuple*)
DBLink **db*;
int *relid*;
Tuple **tuple*;

DESCRIPTION

DBL_add_tuple() adds a tuple filled with NULL values to the relation *relid* in database *db*. The primary key within the tuple is automatically updated (e.g. *wfid* within the wfdisc relation). The function returns the new tuple index, or -1 if an error occured.

DBL_get_tupple() returns the tuple from position *ituple* from relation *relid* in database *db*. The returned tuple can then be used with the get/put attribute routines to manipulate individual tuple entries. The function returns a pointer to the tuple, or NULL if an error occured.

DBL_put_tupple() returns the tuple to the relation *relid* in database *db*. The function returns 1 on success, or 0 if an error occured.

ARGUMENTS

db A pointer to the database object as returned by **db30_create(3)**.

relid The relation ID for the particular relation within database *db* (see **introcss??(3)** for a list),.

ituple The integer index for a tuple within a database relation. For the *C* interface *ituple* starts at 0; for the *FORTRAN* interface *ituple* starts at 1.

tuple A pointer to a tuple within a database relation. The tuple is one of the arguments required for the get/put attribute routines.

SEE ALSO

introcss30(3), db30_create(3)

NAME

DBL_compose_filename – generate the path for an external file referenced in a table

SYNOPSIS

#include ''DBL_defs.h''
#include ''css30_defs.h''

char ∗**DBL_compose_filename(***rel***,** *dir***,** *dfile***)**
Relation ∗*rel***;**
char ∗*dir***;**
char ∗*dfile***;**

DESCRIPTION

DBL_compose_filename() returns a pointer to the pathname for the external file referenced in the relation described by rel and having directory *dir* and filename *dfile* from that relation. (This may be nontrivial because the relation may not be in the current working directory, hence the dir and dfile may not by themselves define a relative path to the actual file.)

SEE ALSO

css30intro(3)

BUGS AND CAVEATS

returns a pointer to a statically allocated string, which will be overwritten by the next call to DBL_compose_filename.

AUTHOR

Danny Harvey

NAME

DBL_free – free all resources associated with a database

SYNOPSIS

#include ''DBL_defs.h''

void DBL_free(*dbl*)
DBLink **dbl***;**

DESCRIPTION

DBL_free() frees resources associated with the database *dbl*. It releases all resources, so no other calls are needed.

ARGUMENTS

dbl The database pointer *dbl* as returned by **DBL_create(3)**.

SEE ALSO

DBL_create(3)

NAME

DBL_get_attrs – get one or more attributes from a single database tuple

SYNOPSIS

#include ''DBL_defs.h''
#include ''css30_defs.h''

int DBL_get_attrs(*db***,** *relid***,** *ituple***,** *attrid1***,** *value1***, [***attrid2***,** *value2***], ...,** *NULL***)**
DBLink ∗*db***;**
int *relid***;**
int *ituple***;**

DESCRIPTION

DBL_get_attrs() will get one or more attributes from a single tuple of a database relation. Attributes are specified as attribute-value groupings in a variable argument list ala *SunView* or *XView*. A *NULL* argument terminates the variable argument list.

ARGUMENTS

db (i) A pointer to the database object as returned by **db30_create(3)**.

relid (i) The relation ID (see **introcss30(3)** for a list).

ituple (i) The ordinal tuple index (starting at 0) within the relation.

These arguments are followed by any number of attribute-value specifications finally terminated by a *NULL* argument. The list of recognized attribute IDs is given in **introcss30(3)**. Note that each attribute ID must be present in the specified relation or an error will occur. The values are all pointers to data types that must be consistent with the attributes.

RETURN VALUE

0 is returned if an error occured and 1 otherwise.

SEE ALSO

introcss30(3), db30_create(3)

NAME

DBL_get_nrels, DBL_get_ntuples, DBL_exist_db, DBL_exist_rel, DBL_get_relid – database utility functions

SYNOPSIS

#include ''DBL_defs.h''

int DBL_get_nrels(*dbl*)
DBLink ∗*dbl*;

int DBL_get_ntuples(*dbl*, *relid*)
DBLink ∗*dbl*;
int *relid*;

int DBL_exist_db(*dbl*)
DBLink ∗*dbl*;

int DBL_exist_rel(*dbl*, *relname*)
DBLink ∗*dbl*;
char ∗*relname*;

int DBL_get_relid(*dbl*, *relname*)
DBLink ∗*dbl*;
char ∗*relname*;

DESCRIPTION

DBL_get_nrels() returns the number of relations (tables) within the database *dbl*. The return value is the number of relations actually contained in the database, and is not determined from the schema description

DBL_get_ntuples() returns the number of tuples (rows) within the relation *relid* in database *dbl*.

DBL_exist_db() returns true if the database *dbl* exists, and returns 0 if it does not. Existence is defined by the database containing a non-empty relation.

DBL_exist_rel() returns true if the relation *relname* exists within the database *dbl*. Existence is defined by the database containing the non-empty relation *relname*.

DBL_get_relid() returns the relation id associated with *relname* within database *dbl*. The relation id is required for other routines such as **DBL_get_attr(3)** and **DBL_put_attr(3)**. The relation id can also be specified symbolically, see **introcss(3)** for a listing.

ARGUMENTS

dbl A pointer to the database as returned by **DBL_create(3)**.

relid A unique relation identifier for a relation (table) within database *dbl*.

relname A string name identifying the relation (table) within a database *dbl*.

SEE ALSO

DBL_create(3), DBL_free(3)

NAME

DBL_nextid – return a unique id for a field in a CSS table

SYNOPSIS

int DBL_nextid(*name*)
char **name*;

DESCRIPTION

This routine returns the next available id for a particular field (attribute) *name* by consulting the *CSS lastid* table, in particular ‘‘master.lastid’’. There should be only one such table on a system, and it must be used to be kept up to date. If programs make up new id’s without updating this table, conflicting id’s will result.

DBL_nextid() opens the table ‘‘master.lastid’’ and looks for a row where the keyname is the same as the passed *name*. If this row is found, that record is locked, and the corresponding keyvalue is incremented. The record is unlocked, and that keyvalue returned.

If no corresponding record is found, or the *lastid* table can’t be opened, the routine returns *-1*.

FILES

master.lastid -- this will be searched for along *JSPC_DBPATH*.

EXAMPLE

```
nxt = DBL_nextid(‘‘wfid’’);
```

AUTHOR

Daniel Quinlan

NAME

DBL_put_attrs – put one or more attributes into a single database tuple

SYNOPSIS

#include ''DBL_defs.h''
#include ''css30_defs.h''

int DBL_put_attrs (*db***,** *relid***,** *ituple***,** *attrid1***,** *value1***, [***attrid2***,** *value2***], ...,** *NULL***)**
DBLink ∗*db***;**
int *relid***;**
int *ituple***;**

DESCRIPTION

DBL_put_attrs() will put one or more attributes into a single tuple of a database relation. Attributes are specified as attribute-value groupings in a variable argument list ala *SunView* or *XView*. A NULL argument terminates the variable argument list.

ARGUMENTS

db (i) A pointer to the database object as returned by **db30_create(3)**.

relid (i) The relation ID (see **introcss(3)** for a list).

ituple (i) The ordinal tuple index (starting at 0) within the relation.

These arguments are followed by any number of attribute-value specifications finally terminated by a *NULL* argument. The list of recognized attribute IDs is given in **introcss30(3)**. Note that each attribute ID must be present in the specified relation or an error will occur. The values are all data types that must be consistent with the attributes.

RETURN VALUE

0 is returned if an error occured and 1 otherwise.

SEE ALSO

introcss30(3), db30_create(3)

NAME

doy2mday, mday2doy, h2e, e2h – conversion of time between month/day, day of year, and epoch time

SYNOPSIS

void doy2mday(*doy*, *year*, *month*, *day*)
int *doy*;
int *year*;
int **month*;
int **day*;

int mday2doy(*year*, *month*, *day*)
int *year*;
int *month*;
int *day*;

double h2e(*year*, *day*, *hour*, *min*, *sec*)
int *year*;
int *day*;
int *hour*;
int *min*;
double *sec* ;

void e2h(*epoch*, *iyear*, *iday*, *ihour*, *imin*, *sec*)
double *epoch*;
int **iyear*;
int **iday*;
int **ihour*;
int **imin*;
double **sec*;

DESCRIPTION

These routines perform conversions between different time and date formats. **doy2mday()** converts from day of year, *doy*, and *year*, to *month* and *day*. **mday2doy()** returns the day of the year given *year*, *month*, and *day*. **h2e()** returns the *epoch* time given *year*, *day*, *hour*, *minute* and *seconds*. **e2h()** converts from *epoch* time to *year*, *day*, *hour*, *minute* and *seconds*.

BUGS AND CAVEATS

No checking is performed on the arguments, so that surprising values for *month*, *day*, *hours* and *seconds* are accepted and return real *epoch* times.

NAME

find_path, find_file_path – search for CSS table or file along JSPC_DBPATH

SYNOPSIS

int find_path(*databasepath***,** *table***,** *tblpath***)**
char ∗*databasepath***;**
char ∗*table***;**
char ∗*tblpath***;**

DESCRIPTION

This routine allows a *CSS* database to be spread out across several directories, rather than in a single directory. This is especially convenient for the ‘‘lookup’’ tables, which are fairly static and may be relevant for a large number of waveform data traces which are more conveniently divided among multiple directories rather than all placed in a single directory.

Find_path searches for a *CSS* database *table* (represented by a flat *ASCII* file) by looking in all the directories specified by the J*SPC_DBPATH* environment variable. The logic follows these steps, stopping when a physical file is found:

1) Check the name composed by appending ‘‘.’’ and *table* to *databasepath*.

2) If *databasepath* contains ‘‘/’’, make a database name by extracting the basename from *databasepath* (whatever follows the last ‘‘/’’ in *databasepath*). If *databasepath* is not an absolute pathname, just use it directly as the database name.

3) Now, for each element of the *JSPC_DBPATH*, check:
   a) the name composed by concatenating the *JSPC_DBPATH* element, ‘‘/’’, database name, ‘‘.’’, and *table*
   b) the name composed by concatenating the *JSPC_DBPATH* element, ‘‘.’’, and *table*.

4) If still not found, return -1, and set *tblpath* to *databasepath.table*. If the file is found, return *0* and set *tblpath* to the correct path.

EXAMPLE

In **find_path**, if *JSPC_DBPATH* were ‘‘/nrdc/wf:/nrdc/station/nrdc’’, and *databasepath* were ‘‘/tmp/example’’ and *table* were ‘‘sensor’’, the following pathnames would be tested, returning *0* with *tblpath* set to the corresponding file at the first accessible file:

/tmp/example.sensor
/nrdc/wf/example.sensor
/nrdc/wf.sensor
/nrdc/station/nrdc/example.sensor
/nrdc/station/nrdc.sensor

If none of these files were found, *tblpath* would be set to ‘‘/tmp/example.sensor’’ and *-1* would be returned.

DIAGNOSTICS

No JSPC_DBPATH in environment!

JSPC_DBPATH must be set to use this routine, although it’s acceptable to have it set to null.

BUGS AND CAVEATS

While the *JSPC_DBPATH* makes it convenient to use database spread out across different directories, it also allows some confusion if databases don’t have unique names.

AUTHOR

Daniel Quinlan

NAME

get_argument, get_directory, get_input_file, get_output_file, last_input_file, last_output_file, next_file, readable_file, writable_file – Standard command line parsing routines

SYNOPSIS

```
#include ''tmpl.h''

char get_option(argc, argv, option_list, num_options)
int             *argc;
char            **argv;
option_t        *option_list;
int             num_options;

char *get_argument()

char *get_directory()

char *get_input_file(input_file)
FILE            **input_file;

char *get_output_file(output_file)
FILE            **output_file;

char *last_input_file(input_file)
FILE            **input_file;

char *last_output_file(output_file)
FILE            **output_file;

FILE *next_file(old_file, new_name, access)
FILE            *old_file;
char            *new_name;
char            *access;

int readable_file(file_name)
char            *file_name;

int writable_file(file_name)
char            *file_name;

typedef struct
{
  char          terse;
  char          *verbose;
  int           arg;
} option_t;
```

DESCRIPTION

These routines are designed to allow the user to easily develop a sophisticated command interface for general applications. Facilities include command parsing for terse and verbose options, command syntax checking, and argument access functions which provide testing for directories and file access. The file **template.c** provides an example for how to use these functions, and is a good starting point for building new applications.

This package is an attempt to be both *POSIX* compliant and user friendly. Every option can be used in verbose mode: **-option** or terse mode: **-o** which are functionally equivalent. Any option accepting an argument must *always* be provided with an argument. Or as was so eloquently expresed in the *POSIX* standard ''option arguments are not optional.'' Finally, for options that don't require an argument, the terse forms can be combined behind a single ''-'' in the normal UN*X fashion.

For example the following calls are equivalent:

```
foo -a -b -c -input input-file -output output-file
foo -abc -i input-file -o output-file
```

The legal options are specified in an **option_t** array. Each element in this array describes one option. An option is specified in three parts: **terse** which is the single character description, **verbose** which is the string description, and **arg** which specifies whether the option requires an argument. The legal values for **arg** are **OPTION_ARG** meaning the option requires an argument, and **NO_OPTION_ARG** meaning it does not.

When **get_option()** is called the command arguments are scanned until a option is found and the associated *terse* character is returned. Options are identified by starting with a ''-''. Note that the ''-'' is *not* included in the **option_t** descriptions. If the **verbose** form is used, it is translated to the **terse** form. If any errors occur parsing the options an illegal character is returned. It is suggested that a *default:* or *else* is provided to catch any unexpected return values. If the option requires an argument the argument must be retrieved using one of the **get_*()** routines below. Failure to **get_*()** a required argument causes an error. Likewise, and error is generated if **get_*()** is called when an argument is not expected. When all options have been retrieved, or the special option ''--'' is reached, **get_option()** returns *NULL*. The ''--'' option is used to pass arguments to the program that would otherwise be identified as options.

It is important that **get_option()** be called repeatedly until all options are processed, since **get_option()** modifies *argc* and *argv* variables as it proceeds. Once all options have been retrieved *argc* and *argv* will be reset so they contain any remaining arguments in the normal way. Note that *argc* and *argv* should not be modified by the user, or unpredictable results may occur.

After an option requiring an argument is found, the argument can be retrieved by calling one of the **get_*()** functions below. Each successive call of **get_*()** returns another value until the next option is found, at which point **get_*()** returns *NULL*. If the special option ''--'' was reached, then all values will be returned. All of the following routines, except where noted, can be called whenever an argument is legal: after an option requiring an argument, or at the end of the option list.

Function **get_argument()** simply returns the next argument, as long as it is not an option, (it does not begin with ''-''). If the special option, ''--'', has been specified then all arguments can be retrieved.

Function **get_directory()** returns the next argument so long as it specifies a valid directory, otherwise it returns NULL.

Function **get_input_file()** returns the next argument so long as it specifies a legal input file, otherwise it returns NULL. The file will be opened for input at the same time.

Function **get_output_file()** returns the next argument so long as it specifies a legal output file, otherwise it returns NULL. The file will be opened for output at the same time.

Function **next_file()** closes the previous open file and opens the next argument for reading / writing. This provides a simple way to process a series of files in a sequential order. Note that the only access modes currently supported are ''r'' and ''w''.

Functions **last_input_file()** and **last_output_file()** will retrieve the last argument from the list and open it for reading / writing. This is useful for cases where the last file has special significance. Note that these functions should only be called after **get_option()** has returned null.

Functions **readable_file()** and **writable_file()** return 1 if their argument is a ''normal'' file, (ie not a directory), and if it can be opened for reading / writing. Note that the file remains closed.

EXAMPLE

Here is an example that, while worthless, illustrates the possibilities.

```
static option_t option_list[] =
{
  'a', ''a-option'', NO_OPTION_ARG,
  'b', ''b-option'', NO_OPTION_ARG,
  'c', ''c-option'', OPTION_ARG
};
#define NUM_OPTIONS (sizeof(option_list) / sizeof(option_t))

main (argc, argv)
int      argc;
char  **argv;
{
   while ((option = get_option(&argc, argv, option_list, NUM_OPTIONS)) != NULL)
      {
         switch (option)
            {
               case ('a'):
                  print(''A-OPTION '');
                  break;
               case ('b'):
                  print(''B-OPTION '');
                  break;
               case ('c'):
                  while ((arg = get_argument()) != NULL)
                     printf(''C-OPTION: %s '', arg);
                  break;
               default:
                  USAGE();
            }
      }
   while ((arg = get_argument()) != NULL)
   printf(''EXTRA: %s '', arg);
}
```

Here is the output produced by a variety of calls

```
shell% foo -a -b-option
A-OPTION B-OPTION

shell% foo -ab
A-OPTION B-OPTION

shell% foo -ab bar snork
A-OPTION B-OPTION EXTRA: bar EXTRA: snork

shell% foo -c bar snork
C-OPTION: bar C-OPTION: snork

shell% foo -c bar snork -ab
C-OPTION: bar C-OPTION: snork A-OPTION B-OPTION

shell% foo -c bar snork -a-option -b bar snork
C-OPTION: bar C-OPTION: snork A-OPTION B-OPTION EXTRA: bar EXTRA: snork

shell% foo -c bar -- -c bar snork
C-OPTION: bar EXTRA: -c EXTRA: bar EXTRA: snork
```

FILES

template.c	Standard template using these routines

SEE ALSO

template(1)

AUTHOR

Eric Winkelman
University of Colorado – Physics department
ewink@lemond.Colorado.EDU

NAME

get_wfdisc30_trace_data – read trace data for CSS 3.0 wfdisc table

SYNOPSIS

int get_wfdisc30_trace_data(*wfdisc_rel*, *i_tuple*, *tstart*, *tend*, *badval*, *nbad*, *tstart_seg*, *dt_seg*, *nsamps_seg*, *data_seg*)
Relation **wfdisc_rel*;
int *i_tuple*;
double *tstart*;
double *tend*;
unsigned int *badval*;
int **nbad*;
double **tstart_seg*;
double **dt_seg*;
int **nsamps_seg*;
float ***data_seg*;

DESCRIPTION

Get_wfdisc30_trace_data() reads the trace data from the disk file referenced in a wfdisc table entry, converting the data to float and multiplying by calib from the same wfdisc entry. If the data is present as a compressed file, the data will be automatically uncompressed. (The data file, however, is left untouched in compressed format).

Get_wfdisc30_trace_data() takes as input a pointer to the wfdisc relation, *wfdisc_rel*, the tuple number *i_tuple*, and the requested starting and ending time *tstart* and *tend* (in epoch time). It malloc's space and returns the start time *tstart_seg*, time increment *dt_seg*, number of samples *nsamps_seg* and a pointer (in *data_seg*) to the relevant portion of the available data. The "relevant" portion of the available data is the data samples from the file for which time is between *tstart* and *tend*. If *tend* <= *tstart*, all the data in the data file is returned.

The badval and nbad parameters are currently disabled.

RETURN VALUES

Get_wfdisc30_trace_data returns 0 in the event of an error, 1 otherwise.

FILES

The file which is referenced in the *i_tuple* entry in the wfdisc relation is read. It may be in compressed format, in which case a temporary uncompressed copy of the data file is made in the same directory.

DIAGNOSTICS

get_wfdisc30_trace_data: DBL_get_tuple error
DBL_get_tuple was unable to read the referenced record.

get_wfdisc30_trace_data: unknown wfdisc datatype
The data type in the wfdisc entry was not recognized.

SEE ALSO

compress(1)

BUGS AND CAVEATS

/usr/ucb/uncompress is run on compressed data files. There must be space available in the current directory and the user must have write permission in that directory. Finally, compress must be present in /usr/ucb/compress.

Calib is applied to scale the data, but calratio is not. The CSS30 spec implies that calratio should also be applied.

AUTHOR

Danny Harvey

NAME

intro – Station-channel view into seismic data

DESCRIPTION

The Station-Channel view software library (libscv.a) provides a consistent application programmer interface to seismic data that may exist in a number of different forms on the mass media. A single station-channel view can be considered to be the basic site information, including detailed station characteristics, such as latitude, longitude and elevation, detailed channel characteristics, such as orientation and depth, detailed sensor characteristics, such as gains and instrument responses, as well as the waveform data samples for a single channel of seismic data over an arbitrary time period. In addition, phase arrival information is also included in a station-channel view. The view should be thought of by the application programmer as an abstract object with all of the information described above.

The current SCV (station-channel view) implementation works with two different storage forms; the CSS v. 2.8 and 3.0 relational flat file forms. Future releases will work with SEED, SEGY, SAC and other data forms. The intent is to provide a consistent programmer interface to the data without dealing with the detailed low level data formats.

LOAD SYNTAX

Application programs using this library should use the following load syntex:

-lscv -lcss28 -lcss30 -ldbl

LIST OF LIBRARY FUNCTIONS (C INTERFACE)

Name	Appears on Page	Description
SCV_create	**SCV_create**(3)	create Station-channel views from disk
SCV_get	**SCV_get**(3)	get Station-channel view attributes
SCV_sift_stachan	**SCV_sift_stachan**(3)	sift Station-channel views
SCV_free_all	**SCV_free**(3)	free Station-channel view resources
SCV_free	**SCV_free**(3)	free Station-channel view resources
SCV_read_wfs	**SCV_read_wfs**(3)	read Station-channel view waveforms
SCV_get_trace	**SCV_get_trace**(3)	get Station-channel view data trace
SCV_free_trace	**SCV_get_trace**(3)	free Station-channel view waveform memory
SCV_get_segment	**SCV_get_segment**(3)	get Station-channel view segment attributes
SCV_get_arrival	**SCV_get_arrival**(3)	get Station-channel view arrival attributes
ARR_get	**SCV_get_arrival**(3)	get Station-channel view arrival attributes
SCV_add_arrival	**SCV_add_arrival**(3)	add a new Station-channel view arrival
SCV_put_arrival	**SCV_put_arrival**(3)	put Station-channel view arrival attributes
ARR_put	**SCV_put_arrival**(3)	put Station-channel view arrival attributes

LIST OF LIBRARY FUNCTIONS (FORTRAN INTERFACE)

Name	Appears on Page	Description
scvcreate	**scvcreate**(3)	create Station-channel views from disk
scvget	**scvget**(3)	get Station-channel view attributes
scvgetarrival	**scvgetarrival**(3)	get Station-channel view arrival attributes
scvgettrace	**scvgettrace**(3)	get Station-channel view data trace
scvselect	**scvselect**(3)	select Station-channel views
scvreadwfs	**scvreadwfs**(3)	read Station-channel view waveforms
scvfreewfs	**scvreadwfs**(3)	free Station-channel view waveforms

LIST OF SCV ATTRIBUTES

Name	Description
SCV_STA	Station code
SCV_CHAN	Channel code
SCV_CHANA	Channel alias code
SCV_TSTART	Start time
SCV_TEND	End time

SCV_LAT	Latitude
SCV_LON	Longitude
SCV_ELEV	Elevation
SCV_DEPTH	Depth
SCV_DEAST	Relative east location
SCV_DNORTH	Relative north location
SCV_DELEV	Relative elevation
SCV_HANG	Horizontal angle of sensor
SCV_VANG	Vertical angle of sensor
SCV_NSEGS	Number of waveform segments
SCV_NARRS	Number of arrivals
SCV_SEG_TSTART	Segment start time
SCV_SEG_TEND	Segment start time
SCV_SEG_DT	Segment time increment
SCV_SEG_NSAMP	Segment number of samples
SCV_SEG_DATA	Segment data buffer
SCV_ARR_TIME	Arrival time
SCV_ARR_PHASE	Arrival phase code
SCV_ARR_AUTH	Arrival author
SCV_ARR_REMARK	Arrival remark
SCV_ARR_FM	Arrival first motion code
SCV_ARR_AMP	Arrival amplitude
SCV_ARR_DELTIM	Arrival time uncertainty

NAME

introcss30 – Utilities for manipulating CSS v. 3.0 flat file databases

DESCRIPTION

The CSS v. 3.0 flat file database software library (libcss30.a) provides a consistent application programmer interface to seismic data that is represented in the Center for Seismic Studies (CSS) v. 3.0 relational schema and is implemented as a series of flat files. These proceedures insulate the user from the details of file I/O and provide easy-to-use methods for getting data from a database and putting data into the database. Note that this library is basically an I/O interface and does not provide most of the DBMS functionality found in a traditional DBMS product. Both C and FORTRAN interfaces are provided.

DATA DEFINITIONS AND STANDARDS

The CSS v. 3.0 schema and the standard ASCII flat file formats are described in the CSS Technical Report C90-01. This library conforms to the v. 3.0 schema and ASCII standard file formats and also provides extensions to the standard for locally developed relations. In addition, this library provides an interface to binary (compiled) versions of the flat files that are C-structure images of the data which are more efficient than the ASCII versions both in terms of file sizes and I/O processing requirements.

The data is organized in the following (approximately) hierarchical manner.

database A database is represented externally as a set of files on disk with a particular file naming convention. Each relation is represented by a separate file and the file names contain the relation name as a suffix in a *.relname* manner. Thus the files *foo.wfdisc, foo.arrival, foo.origin* would define the *wfdisc, arrival* and *origin* relations in a database with the name *foo*. The database name is one of the relation file names without the *.relname* suffix. Calls to **db30_create(3)** or **db30create(3F)** will create an internal representation of a database and map the internal representation to the external file representation.

database object A database object is the internal program representation of a single external database. Each call to **db30_create(3)** or **db30create(3F)** will create and load a new database object and the objects will be independent of each other. An object pointer is returned by **db30_create(3)** and **db30create(3F)** and this pointer is used to reference a particular database in subsequent library calls. Multiple database instances and schema within a single program are supported.

relation A relation is represented externally as a single file and internally in an opaque manner within database objects. A relation can be thought of as a flat table containing rows and columns. Each row is refered to as a *tuple* and each column is refered to as an *attribute*. The current v. 3.0 schema contains 21 relations and we have added 2 local relations. Relations are refered to within the library proceedures both by name and with integer identifiers. In order to avoid confusion with the v. 2.8 schema we have appended "30" to the ends of all names and IDs. The relation IDs (RELID_XXX) are defined in the appropriate C and FORTRAN include files and are listed at the end of this man page.

tuple A tuple is a single row within a single relation and is refered to by the ordinal row number within the relation starting with 0, for the C interface, or 1, for the FORTRAN interface.

attribute An attribute is a single column within one or more relations. An attribute instance is a single attribute value within a specific tuple of a specific relation. The current v. 3.0 schema contains 138 attributes and we have added 14 local attributes. Attributes are referred to within the library procedures both by name and with integer identifiers. In order to avoid confusion with the v. 2.8 schema we have appended "30" to the ends of all names and IDs. The attribute IDs (ATTRID_XXX) are defined in the appropriate C and FORTRAN include files

and are listed at the end of this man page.

EXAMPLE

Following is an example FORTRAN program that opens a database and writes out 20 new tuples into the *wfdisc* relation.

```
      program ftest
c
c     include the css30 defines
c
      include '../css30_defs.i'
c
      character*80 dbname, sta, chan
      real*8 time
c
      if (iargc() .ne. 1) then
          print *,'usage: ftest dbname'
          stop
      end if
      call getarg (1, dbname)
c
c     create and/or open the css 3.0 database
c
      call db30create (dbname, idb)
      if (idb .eq. 0) then
          print *,'ftest: error return from db30create.'
          stop
      end if
      do 10  i = 1, 20
c
c     Make a new tuple
c
          call dbaddtuple (idb, RELID_WFDISC30, ituple)
          if (ituple .lt. 0) then
            print *,'ftest: error return from dbaddtuple.'
            stop
          end if
c
c     Fill in the attributes
c
          write (sta, '(a,i2.2)') 'tst',i
          chan = '00'
          time = 1000000.0 + i*200.0
          call dbputattrs (idb, RELID_WFDISC, ituple,
     +                                ATTRID_STA, sta,
     +                                ATTRID_CHAN, chan,
     +                                ATTRID_TIME, time,
     +                                0)
     10 continue
        stop
        end
```

LOAD SYNTAX

Application programs using this library should use the following load syntex:

-lcss30 -ldbl

LIST OF LIBRARY FUNCTIONS (C INTERFACE)

Name	Appears on Page	Description
db30_create	**db30_create**(3)	create a CSS v. 3.0 database object
DBL_get_tuple	**DBL_get_tuple**(3)	get a tuple from a relation in the database
DBL_get_attrs	**DBL_get_attrs**(3)	get one or more attributes from a tuple in the database
DBL_add_tuple	**DBL_add_tuple**(3)	add a new NULL tuple to a relation in the database
DBL_put_attrs	**DBL_put_attrs**(3)	put one or more attributes into a tuple in the database

LIST OF LIBRARY FUNCTIONS (FORTRAN INTERFACE)

Name	Appears on Page	Description
db30create	**db30create**(3F)	create a CSS v. 3.0 database object
dbgetattrs	**dbgetattrs**(3F)	get one or more attributes from a tuple in the database
dbgetntuples	**dbgetattrs**(3F)	get the number of tuples in a relation
dbaddtuple	**dbaddtuple**(3F)	add a new NULL tuple to a relation in the database
dbputattrs	**dbputattrs**(3F)	put one or more attributes into a tuple in the database

LIST OF RELATION IDS

Name	Description
RELID_AFFILIATION	Affiliation relation ID.
RELID_ARRIVAL	Arrival relation ID.
RELID_ASSOC	Assoc relation ID.
RELID_EVENT	Event relation ID.
RELID_GREGION	Gregion relation ID.
RELID_INSTRUMENT	Instrument relation ID.
RELID_LASTID	Lastid relation ID.
RELID_NETMAG	Netmag relation ID.
RELID_NETWORK	Network relation ID.
RELID_ORIGERR	Origerr relation ID.
RELID_ORIGIN	Origin relation ID.
RELID_REMARK	Remark relation ID.
RELID_SENSOR	Sensor relation ID.
RELID_SITE	Site relation ID.
RELID_SITECHAN	Sitechan relation ID.
RELID_SREGION	Sregion relation ID.
RELID_STAMAG	Stamag relation ID.
RELID_STASSOC	Stassoc relation ID.
RELID_WFDISC	Wfdisc relation ID.
RELID_WFTAG	Wftag relation ID.
RELID_WFTAPE	Wftape relation ID.
RELID_ALIAS	Alias relation ID.
RELID_WFTAR	Wftar relation ID.

LIST OF ATTRIBUTE IDS

Name	Description
ATTRID_ALGORITHM	algorithm attribute ID.
ATTRID_AMP	amp attribute ID.
ATTRID_ARID	arid attribute ID.
ATTRID_AUTH	auth attribute ID.
ATTRID_AZDEF	azdef attribute ID.
ATTRID_AZIMUTH	azimuth attribute ID.

ATTRID_AZRES	azres attribute ID.
ATTRID_BAND	band attribute ID.
ATTRID_BELIEF	belief attribute ID.
ATTRID_CALIB	calib attribute ID.
ATTRID_CALPER	calper attribute ID.
ATTRID_CALRATIO	calratio attribute ID.
ATTRID_CHAN	chan attribute ID.
ATTRID_CHANID	chanid attribute ID.
ATTRID_CLIP	clip attribute ID.
ATTRID_COMMID	commid attribute ID.
ATTRID_CONF	conf attribute ID.
ATTRID_CTYPE	ctype attribute ID.
ATTRID_DATATYPE	datatype attribute ID.
ATTRID_DEAST	deast attribute ID.
ATTRID_DELAZ	delaz attribute ID.
ATTRID_DELSLO	delslo attribute ID.
ATTRID_DELTA	delta attribute ID.
ATTRID_DELTIM	deltim attribute ID.
ATTRID_DEPDP	depdp attribute ID.
ATTRID_DEPTH	depth attribute ID.
ATTRID_DESCRIP	descrip attribute ID.
ATTRID_DFILE	dfile attribute ID.
ATTRID_DIGITAL	digital attribute ID.
ATTRID_DIR	dir attribute ID.
ATTRID_DIST	dist attribute ID.
ATTRID_DNORTH	dnorth attribute ID.
ATTRID_DTYPE	dtype attribute ID.
ATTRID_EDEPTH	edepth attribute ID.
ATTRID_ELEV	elev attribute ID.
ATTRID_EMA	ema attribute ID.
ATTRID_EMARES	emares attribute ID.
ATTRID_ENDTIME	endtime attribute ID.
ATTRID_ESAZ	esaz attribute ID.
ATTRID_ETYPE	etype attribute ID.
ATTRID_EVID	evid attribute ID.
ATTRID_EVNAME	evname attribute ID.
ATTRID_FM	fm attribute ID.
ATTRID_FOFF	foff attribute ID.
ATTRID_GRN	grn attribute ID.
ATTRID_GRNAME	grname attribute ID.
ATTRID_HANG	hang attribute ID.
ATTRID_IDTYPE	idtype attribute ID.
ATTRID_IDVALUE	idvalue attribute ID.
ATTRID_IMB	imb attribute ID.
ATTRID_IML	iml attribute ID.
ATTRID_IMS	ims attribute ID.
ATTRID_INID	inid attribute ID.
ATTRID_INSNAME	insname attribute ID.
ATTRID_INSTANT	instant attribute ID.
ATTRID_INSTYPE	instype attribute ID.
ATTRID_IPHASE	iphase attribute ID.
ATTRID_JDATE	jdate attribute ID.
ATTRID_LAT	lat attribute ID.

ATTRID_LDDATE	lddate attribute ID.
ATTRID_LINENO	lineno attribute ID.
ATTRID_LOCATION	location attribute ID.
ATTRID_LOGAT	logat attribute ID.
ATTRID_LON	lon attribute ID.
ATTRID_MAGID	magid attribute ID.
ATTRID_MAGNITUDE	magnitude attribute ID.
ATTRID_MAGTYPE	magtype attribute ID.
ATTRID_MB	mb attribute ID.
ATTRID_MBID	mbid attribute ID.
ATTRID_ML	ml attribute ID.
ATTRID_MLID	mlid attribute ID.
ATTRID_MS	ms attribute ID.
ATTRID_MSID	msid attribute ID.
ATTRID_NASS	nass attribute ID.
ATTRID_NCALIB	ncalib attribute ID.
ATTRID_NCALPER	ncalper attribute ID.
ATTRID_NDEF	ndef attribute ID.
ATTRID_NDP	ndp attribute ID.
ATTRID_NET	net attribute ID.
ATTRID_NETNAME	netname attribute ID.
ATTRID_NETTYPE	nettype attribute ID.
ATTRID_NSAMP	nsamp attribute ID.
ATTRID_NSTA	nsta attribute ID.
ATTRID_OFFDATE	offdate attribute ID.
ATTRID_ONDATE	ondate attribute ID.
ATTRID_ORID	orid attribute ID.
ATTRID_PER	per attribute ID.
ATTRID_PHASE	phase attribute ID.
ATTRID_PREFOR	prefor attribute ID.
ATTRID_QUAL	qual attribute ID.
ATTRID_RECT	rect attribute ID.
ATTRID_REFSTA	refsta attribute ID.
ATTRID_REMARK	remark attribute ID.
ATTRID_RSPTYPE	rsptype attribute ID.
ATTRID_SAMPRATE	samprate attribute ID.
ATTRID_SDEPTH	sdepth attribute ID.
ATTRID_SDOBS	sdobs attribute ID.
ATTRID_SEAZ	seaz attribute ID.
ATTRID_SEGTYPE	segtype attribute ID.
ATTRID_SLODEF	slodef attribute ID.
ATTRID_SLORES	slores attribute ID.
ATTRID_SLOW	slow attribute ID.
ATTRID_SMAJAX	smajax attribute ID.
ATTRID_SMINAX	sminax attribute ID.
ATTRID_SNR	snr attribute ID.
ATTRID_SRN	srn attribute ID.
ATTRID_SRNAME	srname attribute ID.
ATTRID_STA	sta attribute ID.
ATTRID_STANAME	staname attribute ID.
ATTRID_STASSID	stassid attribute ID.
ATTRID_STATYPE	statype attribute ID.
ATTRID_STIME	stime attribute ID.

ATTRID_STRIKE	strike attribute ID.
ATTRID_STT	stt attribute ID.
ATTRID_STX	stx attribute ID.
ATTRID_STY	sty attribute ID.
ATTRID_STYPE	stype attribute ID.
ATTRID_STZ	stz attribute ID.
ATTRID_SXX	sxx attribute ID.
ATTRID_SXY	sxy attribute ID.
ATTRID_SXZ	sxz attribute ID.
ATTRID_SYY	syy attribute ID.
ATTRID_SYZ	syz attribute ID.
ATTRID_SZZ	szz attribute ID.
ATTRID_TAGID	tagid attribute ID.
ATTRID_TAGNAME	tagname attribute ID.
ATTRID_TAPEBLOCK	tapeblock attribute ID.
ATTRID_TAPEFILE	tapefile attribute ID.
ATTRID_TIME	time attribute ID.
ATTRID_TIMEDEF	timedef attribute ID.
ATTRID_TIMERES	timeres attribute ID.
ATTRID_TSHIFT	tshift attribute ID.
ATTRID_UNCERTAINTY	uncertainty attribute ID.
ATTRID_VANG	vang attribute ID.
ATTRID_VMODEL	vmodel attribute ID.
ATTRID_VOLNAME	volname attribute ID.
ATTRID_WFID	wfid attribute ID.
ATTRID_WGT	wgt attribute ID.
ATTRID_ARRAY	array attribute ID.
ATTRID_BLKFAC	blkfac attribute ID.
ATTRID_CHANA	chana attribute ID.
ATTRID_CHANO	chano attribute ID.
ATTRID_COMPA	compa attribute ID.
ATTRID_DESCR	descr attribute ID.
ATTRID_NBYTE	nbyte attribute ID.
ATTRID_NETWRK	netwrk attribute ID.
ATTRID_STAA	staa attribute ID.
ATTRID_STAO	stao attribute ID.
ATTRID_TARNAM	tarnam attribute ID.
ATTRID_TCALIB	tcalib attribute ID.
ATTRID_TFILE	tfile attribute ID.
ATTRID_TPFILE	tpfile attribute ID.

NAME

read_response, write_response, eval_response, free_response – manipulate response files

SYNOPSIS

#include ''response.h''

int read_response(*file***,** *p_response***)**
FILE ∗*file***;**
Response∗∗*p_response***;**

int write_response(*file***,** *response***)**
FILE ∗*file***;**
Response ∗*response***;**

int eval_response(*omega***,** *response***,** *real***,** *imag***)**
double *omega***;**
Response ∗*response***;**
double ∗*real***;**
double ∗*imag***;**

int free_response(*response***)**
Response ∗*response***;**

DESCRIPTION

These routines read and write response files in the format suggested by CSS (see response(5)), and evaluate the response function at a particular frequency.

Read_response() reads response from the file pointed to by the open file descriptor, and returns a pointer to a structure which represents the information from the file. (Any memory needed by this structure was allocated by the routine read_response using malloc).

Write_response() writes out the response represented by the passed structure into the file pointed to by the open file descriptor. The output format follows the CSS suggested format response(5).

Eval_response() returns the result of evaluating the response function at the specified frequency omega (in radians/sec). This complex response is **NOT** normalized; it must be calibrated by using the CALIB, CALRATIO, and CALPER fields from the wfdisc and sensor relations.

Free_response() frees the memory associated with the response structure.

RETURN VALUES

These routines return 0 on success, non-zero in the event of failure. In the event of an error, more verbose error messages are logged using register_error(3) and may be printed using clear_register(3).

FILES

The response file must follow the format defined in response(5).

EXAMPLE

```
Response ∗response ;
double omega, real, imag ;
FILE ∗file, ∗out ;

file = fopen ( argv[1], ''r'' ) ;
if ( read_response ( file, &response ) != 0 )
    {
    clear_register ( 1 ) ;
    exit (1) ;
    }
fclose ( file ) ;

if ( write_response ( stdout, response ) != 0 )
    {
```

```
    clear_register (1);
    exit ( 1) ;
    }
fclose ( out ) ;

for ( omega = 1.0 ; omega < 100.0 * 2.0 * 3.14159 ; omega += 50.0 )
    {
    if ( eval_response ( omega, response, &real, &imag ) != 0 )
        {
        clear_register ( 1 ) ;
        exit (1) ;
        }
    printf ( ''%10.2f    %10.3f  %10.3f\n'', omega, real, imag ) ;
    }

free_response ( response ) ;
```

DIAGNOSTICS

Bad source
: source field must be either ''theoretical'' or ''measured''

Unknown filter type
: only ''fir'', ''fap'' and ''paz'' filter types are allowed (and only ''paz'' is supported).

SEE ALSO

response(5)
error(3)

BUGS AND CAVEATS

Only the ''paz'' (poles and zeros) response type is currently supported.

AUTHOR

Daniel Quinlan

NAME

SCV_add_arrival – add a new arrival to the station-channel view

SYNOPSIS

#include ''scv.h''

SCVArrival ∗**SCV_add_arrival**(*scv*, *time*)
SCV ∗*scv*;
double *time*;

DESCRIPTION

The procedure **SCV_add_arrival()** adds a new arrival to the station-channel view (*scv*). A new, empty, arrival is inserted at *time*, and the arrivals index is returned. Arrivals are kept in *time* order, with the first arrival having index 0.

Note that as arrivals are added, the index of existing arrivals can change.

ARGUMENTS

scv A pointer to the Station-Channel View structure as returned by **SCV_create(3)**.

time The time the arrival occurred in the *scv*.

RETURN VALUE

If **SCV_add_arrival()** fails, then *-1* will be returned and an error message is printed on standard error output. If **SCV_add_arrival()** succeeds, then the index of the new arrival is returned.

DIAGNOSTICS

If an error is discovered, an error message will be printed to stdout, and **SCV_add_arrival()** returns *-1*.

SEE ALSO

introscv(3), SCV_get_arrival(3), SCV_put_arrival(3)

NAME

SCV_create – create one or more Station-Channel Views

SYNOPSIS

#include ''scv.h''

SCV ****SCV_create**(*schema*, *ndbs*, *dbnames*, *stachan_sift*, *tstart_sift*, *tend_sift*, *exist*, *nscvs*)
char **schema;*
int *ndbs;*
char ***dbnames;*
char **stachan_sift;*
char **tstart_sift;*
char **tend_sift;*
int *exist;*
int **nscvs;*

DESCRIPTION

The proceedure **SCV_create()** will create one or more new Station-Channel views from a variety of different external data formats.

ARGUMENTS

schema (i) A pointer to a NULL terminated character string that defines the external data format. The following *schema* strings are currently recognized.
''css2.8'' - CSS v. 2.8 flat file format.
''css3.0'' - CSS v. 3.0 flat file format.

ndbs (i) The number of input names describing the data. Currently this must be one.

dbnames (i) An array of character string pointers with names describing the input data. There should be *ndbs* string pointers in the array. The nature of these names is format (schema) dependent. For flat file relational databases these names correspond to the database names, which are flat file names without the *.relname* suffix. The current version will only use the first name.

stachan_sift (i) A station-channel sift key. This string specifies the subset of station-channel data streams that will be read from the input external data. If this string pointer is NULL, then no station-channel sifting is done. The string format is *<sta>:<chan>[,<sta2>:<chan2>,[...]]* where *<sta>* is the station string and *<chan>* is the channel string. The asterisk (*) character can be used as a wild card in place of *<sta>* or *<chan>* which matches anything. Multiple station-channel specifications are separated by commas (,). Normal UNIX regular expression matching is also used (ala ed). (e.g. A.*:* matches all channels of all stations beginning with A.)

tstart_sift (i) A data sift start time. This string specifies the subset of arrivals and waveform segments that will be read from the input external data. Only arrivals and waveform segments with time later than *tstart_sift* will be read. If this string pointer is NULL, then no start time sifting is done. The start time can be specified in several different ways.
1. A character string epoch time in seconds.
2. A character string in ''year:month:day:hour:minute:second'' form.
3. A character string in ''julianday:hour:minute:second'' form.

tend_sift (i) A data sift end time. This string specifies the subset of arrivals and waveform segments that will be read from the input external data. Only arrivals and waveform segments with time earlier than *tend_sift* will be read. If this string pointer is NULL, then no start time sifting is done. The end time is specified in the same manner as the start time.

exist (i) Waveform existence flag. If this is set to non-zero, then only station-channels with existing waveforms on disk will be read.

nscvs (o) The number of Station-Channel views read from the input external data. If this is 0, then no station-channel streams met the sift requirements.

RETURN VALUE

If **SCV_create()** fails, then NULL will be returned and an error message is (usually) printed on standard error output. If **SCV_create()** succeeds, then a pointer to an array of SCV pointers is returned.

SEE ALSO

introscv(3)

NAME

SCV_free, SCV_free_all – free memory associated with SCV's

SYNOPSIS

#include ''scv.h''

void SCV_free(*scv*)
SCV **scv;*

void SCV_free_all(*scvs*, *nscvs*)
SCV ***scvs;*
int *nscvs;*

DESCRIPTION

SCV_free() will free all memory resources associated with an individual station-channel view (*scv*).

SCV_free_all() will free all memory resources associated with an array of station-channel views. This includes both the individual station-channel view allocations as well as the SCV pointer array.

ARGUMENTS

scv	A single station-channel view.
nscvs	The number of station-channel views.
scvs[nscvs]	The array of input station-channel views.

DIAGNOSTICS

All errors caught will be reported to *stderr*.

SEE ALSO

introscv(3) SCV_create(3)

NAME

SCV_get – get one or more Station-Channel View attributes

SYNOPSIS

#include ''scv.h''

int SCV_get (*scv*, *attr1*, *value1*, [*attr2*, *value2*], ..., *NULL*)
SCV **scv*;

DESCRIPTION

SCV_get() will get one or more Station-Channel View attributes. Attributes are specified as attribute-value groupings in a variable argument list ala SunView or XView.

ARGUMENTS

scv (i) A pointer to the Station-Channel View as returned by **SCV_create(3)**.

This is followed by any number of attribute-value specifications finally terminated by a NULL argument. Following is a list of the attributes and the value types that will be returned.

attribute	value description
SCV_STA	The station code. Type: char **
SCV_CHAN	The channel code. Type: char **
SCV_CHANA	The channel alias code. This is used to define channel codes that are longer and more descriptive than the original data format would allow. Type: char **
SCV_TSTART	The epoch start time. This is the earliest time for all waveform segment samples and arrivals. Type: double *
SCV_TEND	The epoch end time. This is the latest time for all waveform segment samples and arrivals. Type: double *
SCV_LAT	The station latitude in degrees. Type: float *
SCV_LON	The station longitude in degrees. Type: float *
SCV_ELEV	The station elevation above sea level in km. Type: float *
SCV_DEPTH	The channel depth of emplacement in km relative to the station elevation. Type: float *
SCV_DEAST	The station relative east position in km relative to some reference station. Type: float *
SCV_DNORTH	The station relative north position in km relative to some reference station. Type: float *
SCV_DELEV	The station relative elevation in km relative to some reference station. Type: float *

SCV_HANG The channel sensor horizontal angle orientation in degrees. This is measured positive clockwise from north.
Type: float *

SCV_VANG The channel sensor vertical angle orientation in degrees. This is measured positive down from horizontal.
Type: float *

SCV_NSEGS The total number of waveform segments for this station-channel view.
Type: int *

SCV_NARRS The total number of phase arrivals for this station-channel view.
Type: int *

SEE ALSO

introscv(3), SCV_create(3)

NAME

SCV_get_arrival, ARR_get – get attributes from a Station-channel view arrival

SYNOPSIS

#include ''scv.h''

SCVArrival ∗**SCV_get_arrival**(*scv*, *arr*, *attr1*, *value1*, **...** *NULL*)
SCV ∗*scv*;
int *arr*;

SCVArrival ∗**ARR_get**(*arr_ptr*, *attr1*, *value1*, **...** *NULL*)
SCVArrival ∗*arr_ptr*;

DESCRIPTION

The procedures **SCV_get_arrival()** and **ARR_get()** retrieve attributes from an individual station-channel arrival. Attributes are specified as attribute-value pairs in a variable argument list ala *SunView* or *XView*. The list of attribute-value pairs is terminated by a *NULL* value.

ARGUMENTS

scv A pointer to the Station-Channel View structure as returned by **SCV_create(3)**.

arr The index of the arrival in the *scv*, by time of arrival. The first arrival will have an index of 0.

arr_ptr A pointer to the Station-Channel View Arrival structure as returned by previous calls to **SCV_get_arrival()** or **ARR_get()**.

Subsequent arguments consist of any of the following ID/value pairs:

ID	**definition**	**type**
SCV_ARR_TIME	Arrival time	double ∗
SCV_ARR_PHASE	Phase code	char ∗∗

The variable argument list is terminated with: *NULL*

RETURN VALUE

If **SCV_get_arrival()** / **ARR_get()** fails, then *NULL* is returned and an error message is printed on standard error output. If **SCV_get_arrival()** / **ARR_get()** succeeds, then a pointer to the arrival is returned.

DIAGNOSTICS

If an error is discovered, an error message will be printed to stdout, and **SCV_get_arrival()** / **ARR_get()** returns *NULL*

SEE ALSO

introscv(3), SCV_create(3), SCV_get(3)

NAME

SCV_get_segment – get attributes from a Station-channel view segment

SYNOPSIS

#include ''scv.h''

int SCV_get_segment(*scv***,** *seg***,** *attr1***,** *value1***, ...** *NULL***)**
SCV ***scv;**
int *seg;*

DESCRIPTION

The procedure **SCV_get_segment()** retrieves attributes from an individual station-channel view data segment. Attributes are specified as attribute-value pairs in a variable argument list ala *SunView* or *XView*. The list of attribute-value pairs is terminated by a *NULL* value.

ARGUMENTS

scv A pointer to the Station-Channel View structure as returned by **SCV_create(3)**.

seg The index of the segment in the *scv*. The first segment will have an index of 0.

Subsequent arguments consist of any of the following ID/value pairs:

ID	**definition**	**type**
SCV_SEG_EXIST	Existence flag	int *
SCV_SEG_NBAD	Bad value count	int *
SCV_SEG_TSTART	Start time	double *
SCV_SEG_TEND	End time	double *
SCV_SEG_NSAMPS	Number of samples	int *
SCV_SEG_INSTR	Instrument response	SCVInstrument **
SCV_SEG_RESPFILE	Instrument response file	char **
SCV_SEG_DT	Delta Time	double *
SCV_SEG_DATA	Segment data	float **

The variable argument list is terminated with *NULL*

RETURN VALUE

If **SCV_get_segment()** succeeds *(0)* is returned.

DIAGNOSTICS

If an error is discovered an error message is printed on stdout, and **SCV_get_segment()** returns *(-1)*.

SEE ALSO

introscv(3), SCV_create(3), SCV_get(3)

NAME

SCV_get_trace, SCV_free_trace – manipulate data samples for a Station-Channel View

SYNOPSIS

#include ''scv.h''

Trace *SCV_get_trace(*scv***,** *tstart***,** *tend***)**
SCV **scv;*
double *tstart;*
double *tend;*

void SCV_free_trace (*trace*)
Trace **trace;*

DESCRIPTION

The procedure **SCV_get_trace()** will retrieve waveform data samples from an individual station-channel view for a specified start and end time. The data is either read from memory or from disk according to whether or not **SCV_read_wfs(3)** has been previously called. Only data correspnding to the requested times is returned.

SCV_free_trace() will free all memory resources associated with a Trace linked list for an individual Station-Channel View (as returned by **SCV_get_trace()**). Note that this only applies to the temporary memory resources that are allocated when data is read from disk by **SCV_get_trace()** as opposed to the permanent resources that are allocated by **SCV_read_wfs(3)**.

ARGUMENTS

scv	(i) A pointer to the Station-Channel View structure as returned by **SCV_create(3)**.
tstart	(i) The requested epoch start time.
tend	(i) The requested epoch end time.

RETURN VALUE

If **SCV_get_trace()** fails, then NULL will be returned and an error message is (usually) printed on standard error output. If **SCV_get_trace()** succeeds, then a pointer to a linked list of Trace structures is returned. Each element in the linked list corresponds to a single continuous waveform segment. A single Trace structure is defined as follows.

```
typedef struct trace_ {
        SCV *scv;               /* Parent SCV */
        double tstart;          /* Segment epoch start time. */
        double dt;              /* Segment time increment. */
        int nsamps;             /* Segment number of samples. */
        float *data;            /* Segment data samples. */
        float *data_free;
        int data_malloc;
        struct trace_ *prev;    /* Previous linked list element. */
        struct trace_ *next;    /* Next linked list element. */
} Trace;
```

SEE ALSO

introscv(3), SCV_create(3), SCV_read_wfs(3)

NAME

SCV_put_arrival, ARR_put – set attributes for a Station-channel view arrival

SYNOPSIS

#include ''scv.h''

SCVArrival ∗**SCV_put_arrival**(*scv*, *arr*, *attr1*, *value1*, **...** *NULL*)
SCV ∗*scv*;
int *arr*;

SCVArrival ∗**ARR_put**(*arr_ptr*, *attr1*, *value1*, **...** *NULL*)
SCVAarrival ∗*arr_ptr*;

DESCRIPTION

The procedures **SCV_put_arrival** and **SCV_put_arrival** set attribute values in an individual station-channel arrival. Attributes are specified as attribute-value pairs in a variable argument list ala *SunView* or *XView*. The list of attribute-value pairs is terminated by a *NULL* value.

ARGUMENTS

scv A pointer to the Station-Channel View structure as returned by **SCV_create(3)**.

arr The index of the arrival in the *scv*, by time of arrival. The first arrival will have an index of 0.

arr_ptr A pointer to the Station-Channel View Arrival structure as returned by previous calls to **SCV_get_arrival()** or **ARR_get()**.

Subsequent arguments consist of any of the following ID/value pairs:

ID	**definition**	**type**
SCV_ARR_TIME	Arrival time	double
SCV_ARR_PHASE	Phase code	char ∗
SCV_ARR_FM	Arrival first motion	char ∗
SCV_ARR_AMP	Arrival amplitude	double
CV_ARR_DELTIM	Arrival time uncertainty	double
SCV_ARR_AUTH	Arrival author	char ∗
SCV_ARR_REMARK	Arrival remark	char ∗

The variable argument list is terminated with:
NULL

RETURN VALUE

If **SCV_put_arrival()** / **ARR_put()** fails, then *0* is returned and an error message is printed on standard error output. If **SCV_put_arrival()** / **ARR_put()** succeeds, then a non-zero value is returned.

DIAGNOSTICS

If an error is discovered, an error message will be printed to stdout, and **SCV_put_arrival** / **ARR_put()** returns *0*

SEE ALSO

introscv(3), SCV_create(3), SCV_get_arrival(3)

NAME

SCV_read_wfs – read waveform sample data into memory

SYNOPSIS

#include ''scv.h''

int SCV_read_wfs(*scv***)**
SCV ***scv*;**

DESCRIPTION

The proceedure **SCV_read_wfs()** reads into memory all waveform segments for an individual station-channel view. Subsequent calls to **SCV_get_trace(3)** will take the waveform samples from the memory buffers instead of from disk thus improving performance at the expense of memory.

ARGUMENTS

scv (i) A pointer to the Station-Channel View structure as returned by **SCV_create(3)**.

RETURN VALUE

If **SCV_read_wfs()** fails, then 0 will be returned and an error message is (usually) printed on standard error output. If **SCV_read_wfs()** succeeds, then 1 is returned.

SEE ALSO

introscv(3), SCV_create(3), SCV_get_trace(3)

NAME

SCV_sift_stachan – sift a set of station-channel views

SYNOPSIS

#include ''scv.h''

SCV ∗∗**SCV_sift_stachan**(*nscvs*, *scvs*, *stachan_sift*, *nscvs_out*)
int *nscvs*;
SCV ∗∗*scvs*;
char ∗*stachan_sift*;
int ∗*nscvs_out*;

DESCRIPTION

SCV_sift_stachan() sifts a set of station-channel views, according to a specified station-channel sift key, and returns the subset of the station-channel views (*scvs*) that conform to the sifting.

ARGUMENTS

nscvs The number of input station-channel views.

scvs[nscvs] The array of input station-channel views.

stachan_sift A station-channel sift key. This string specifies the subset of station-channel data streams that will be returned. The string format is

'<sta>:<chan>[,<sta2>:<chan2>,[...]]'

where <sta> is the station string and <chan> is the channel string. The asterisk (∗) character can be used as a wild card in place of <sta> or <chan> which matches anything. Multiple station-channel specifications are separated by commas (,). Normal UNIX regular expression matching is also used (ala ed). e.g. A.∗:∗ matches all channels of all stations beginning with A.

nscvs_out The number of sifted station-channel views.

RETURN VALUE

On success **SCV_sift_stachan()** returns a pointer to an array of SCV pointers. On failure **SCV_sift_stachan()** returns NULL. These pointers define the sifted station-channel views. Note that this array of pointers is dynamically allocated here and is distinct from the input pointer array.

DIAGNOSTICS

All errors caught will be reported to stderr.

SEE ALSO

introscv(3) SCV_create(3) SCV_get(3) SCV_free(3)

NAME

time_string2epoch – conversion to epoch time from an ascii string

SYNOPSIS

int time_string2epoch(*string*, *time*)
char ∗*string*;
double ∗*time*;

DESCRIPTION

This routine converts an *ASCII string* into an epoch *time*. The input *string* must be in one of the following formats:

557656335.496
epoch *time* in *ASCII*

1988:feb:15:22:35:43.5
year:month:day:hour:min:sec

1990078:03:10:15.6
juliandate:hour:min:sec

RETURN VALUES

Returns *0* if the string cannot be converted, *1* otherwise.

NAME

db30create – create one or more Station-Channel Views, FORTRAN interface

SYNOPSIS

call db30create (*dbname*, *idb*)
character∗(∗) *dbname*
integer *idb*

DESCRIPTION

The FORTRAN subroutine **db30create()** will create a CSS v. 3.0 database object, map it to the external flat file representation and load it with data from any existing external flat files. This is a FORTRAN version of **db30_create(3)**.

ARGUMENTS

dbname (i) A character string with the external flat file database name. This is the same as one of the relation file names without the *.relname* suffix.

idb (o) The output database object handle. This handle must be used in subsequent css30 library calls. If the **idb** return value is 0, then an error occured and an error message is (usually) printed on standard error output.

SEE ALSO

introcss30(3), db30_create(3)

NAME

dbaddtuple – add a NULL tuple to a relation, FORTRAN version

SYNOPSIS

include '/sw/loc/include/css30_defs.i'

call dbaddtuple(*idb***,** *relid***,** *ituple***)**
integer *idb*
integer *relid*
integer *ituple*

DESCRIPTION

The FORTRAN subroutine **dbaddtuple()** will add a tuple filled with *NULL* values to the end of the tuple list within a relation. The primary key (e.g. *wfid* within the *wfdisc* relation) within the tuple is automatically updated. This is a FORTRAN version of **DBL_add_tuple(3)**.

ARGUMENTS

db (i) The database object handle as returned by **db30create(3F)**.

relid (i) The relation ID (see **introcss30(3)** for a list).

ituple (o) The new tuple index is returned or -1 if an error occured. Note that a return value of 1 is OK and it indicates that a new relation was made.

SEE ALSO

introcss30(3), db30create(3F), DBL_add_tuple(3)

NAME

dbgetattrs – get one or more attributes from a single database tuple, FORTRAN version

SYNOPSIS

include '/sw/loc/include/css30_defs.i'

call dbgetattrs(*idb*, *relid*, *ituple*, *attrid1*, *value1*, [*attrid2*, *value2*], ..., *0*)
integer *idb*
integer *relid*
integer *ituple*

call dbgetntuples (*idb*, *relid*, *ntuples*)
integer *idb*
integer *relid*
integer *ntuples*

DESCRIPTION

The FORTRAN subroutine **dbgetattrs()** will get one or more attributes from a single tuple of a database relation. Attributes are specified as attribute-value groupings in a variable argument list ala *SunView* or *XView*. A *0* argument terminates the variable argument list. This is a FORTRAN version of *DBL_get_attrs(3).*

The FORTRAN subroutine **dbgetntuples()** will return the number of tuples in *ntuples* that currently exist within the specified relation.

ARGUMENTS

idb (i) A database object handle as returned by **db30create(3F)**.

relid (i) The relation ID (see **introcss30(3)** for a list).

ituple (i) The ordinal tuple index (starting at 1) within the relation.

These arguments are followed by any number of attribute-value specifications finally terminated by a *0* argument. The list of recognized attribute IDs is given in **introcss30(3)**. Note that each attribute ID must be present in the specified relation or an error will occur. The values are all data types that must be consistent with the attributes.

SEE ALSO

introcss30(3), db30create(3F), DBL_get_attrs(3)

NAME

dbputattrs – put one or more attributes into a single database tuple, FORTRAN version

SYNOPSIS

include '/sw/loc/include/css30_defs.i'

call dbputattrs(*idb*, *relid*, *ituple*, *attrid1*, *value1*, [*attrid2*, *value2*], ..., *0*)
integer *idb*
integer *relid*
integer *ituple*

DESCRIPTION

The FORTRAN subroutine **dbputattrs()** will put one or more attributes into a single tuple of a database relation. Attributes are specified as attribute-value groupings in a variable argument list ala *SunView* or *XView*. A 0 argument terminates the variable argument list. This is a FORTRAN version of *DBL_put_attrs(3)*.

ARGUMENTS

idb (i) A database object handle as returned by **db30create(3F)**.

relid (i) The relation ID (see **introcss30(3)** for a list).

ituple (i) The ordinal tuple index (starting at 1) within the relation.

These argumenta are followed by any number of attribute-value specifications finally terminated by a *0* argument. The list of recognized attribute IDs is given in **introcss30(3)**. Note that each attribute ID must be present in the specified relation or an error will occur. The values are all data types that must be consistent with the attributes.

SEE ALSO

introcss30(3), db30create(3F), DBL_put_attrs(3)

NAME

doy2mday, mday2doy, h2e, e2h – conversion of time between month/day, day of year, and epoch time

SYNOPSIS

subroutine doy2mday(*doy*, *year*, *month*, *day*)
integer *doy*;
integer *year*;
integer *month*;
integer *day*;

integer function mday2doy(*year*, *month*, *day*)
integer *year*;
integer *month*;
integer *day*;

real*8 function h2e(*year*, *day*, *hour*, *min*, *sec*)
integer *year*;
integer *day*;
integer *hour*;
integer *min*;
real*8 *sec*;

subroutine e2h(*epoch*, *iyear*, *iday*, *ihour*, *imin*, *sec*)
real*8 *epoch*;
integer *iyear*;
integer *iday*;
integer *ihour*;
integer *imin*;
real*8 *sec*;

DESCRIPTION

These routines perform conversions between different time and date formats. **doy2mday()** converts from day of year (*doy*) and *year* to *month* and *day*. **mday2doy()** returns the day of the year (*doy*) given *year*, *month*, and *day*. **h2e()** returns the *epoch* time given *year*, *day*, *hour*, *minute* and *seconds*. **e2h()** converts from *epoch* time to *year*, *day*, *hour*, *minute* and *seconds*.

BUGS AND CAVEATS

No checking is performed on the arguments, so that surprising values for *month*, *day*, *hours* and *seconds* are accepted and return real *epoch* times.

NAME

getwf – read trace data for CSS 3.0 wfdisc table

SYNOPSIS

subroutine getwf(*idbl*, *ituple*, *tstart*, *tend*, *badval*, *nbad*, *tstartseg*, *dtseg*, *nsamps*, *maxsamples*, *data*)
integer *idbl*
integer *ituple*
real*8 *tstart*
real*8 *tend*
integer *badval*
integer *nbad*
real*8 *tstartseg*
real*8 *dtseg*
int *nsamps*
real *data*

DESCRIPTION

Getwf() reads the trace data from the disk file referenced in a wfdisc table entry, converting the data to float and multiplying by calib from the same wfdisc entry. If the data is present as a compressed file, the data will be automatically uncompressed. (The data file, however, is left untouched in compressed format).

Getwf() takes as input a dbl handle *idbl*, the wfdisc tuple number *ituple*, the requested starting and ending time *tstart* and *tend* (in epoch time), and the maximum size of the data array. It returns in *data* the relevant portion of the available data, the start time *tstartseg* for that data, the time increment *dtseg*, and the number of samples *nsamps*. The "relevant" portion of the available data is the data samples from the file for which time is between *tstart* and *tend*. If *tend* <= *tstart*, all the data in the data file is returned.

The *badval* and *nbad* parameters are currently disabled.

FILES

The file which is referenced in the *ituple* entry in the wfdisc relation is read. It may be in compressed format, in which case a temporary uncompressed copy of the data file is made in the same directory.

DIAGNOSTICS

getwf: wfdisc table doesn't exist
: Couldn't open the wfdisc relation.

getwf: tuple index outside of range
: The ituple tuple index was less than one or greater than the number of tuples in the wfdisc relation.

getwf: error return from get_wfdisc30_trace_data
: Some error occurred in get_wfdisc30_trace_data, probably described below.

getwf: Truncating data array
: The passed array was not large enough to hold the entire data segment, so only part of the data is being returned.

get_wfdisc30_trace_data: DBL_get_tuple error
: DBL_get_tuple was unable to read the referenced record.

get_wfdisc30_trace_data: unknown wfdisc datatype
: The data type in the wfdisc entry was not recognized.

SEE ALSO

compress(1)

BUGS AND CAVEATS

/usr/ucb/uncompress is run on compressed data files. There must be space available in the current directory and the user must have write permission in that directory. Finally, compress must be present

in /usr/ucb/compress.

Calib is applied to scale the data, but calratio is not. The CSS30 spec implies that calratio should also be applied.

The variable *ituple* in the FORTRAN routines counts from 1 rather 0. Ie, the first tuple in a relation is tuple #1 in Fortran, and tuple #0 in c.

AUTHOR

Danny Harvey

NAME

putwf – write trace data for CSS 3.0 wfdisc table

SYNOPSIS

subroutine putwf(*idbl*, *ituple*, *data*)
integer *idbl*
integer *ituple*
real *data*

DESCRIPTION

Putwf() writes a data file according to the specifications in entry #*ituple* of the wfdisc table for the database handle *idbl*. The *wfdisc* entry must have been previously filled in with the correct information. In particular, **putwf()** uses *nsamp*, *datatype*, and *calib* from the *wfdisc* tuple. The data is converted from real to the specified data type and divided by calib as it is written out.

FILES

The file which is referenced in the *ituple* entry in the wfdisc relation is written.

DIAGNOSTICS

putwf: wfdisc table doesn't exist.
The wfdisc table couldn't be opened.

putwf: tuple index outside of range.
The ituple tuple index was less than one or greater than the number of tuples in the wfdisc relation.

putwf: Illegal no. of samples.
nsamp in the wfdisc tuple was less than 1

putwf: Error writing file
An error occurred during writing.

putwf: output writing file truncated.
The entire data file was not written.

putwf: Unknown datatype
The data type in the wfdisc entry was not recognized.

SEE ALSO

getwf(3f), db30create(3f), dbaddtuple(3f), dbgetattrs(3f), dbputattrs(3f)

introcss30(3), db30_create(3), DBL_add_tuple(3), DBL_get_attrs(3), DBL_put_attrs(3), get_wfdisc30_trace_data(3)

BUGS AND CAVEATS

Calib is applied to scale the data, but calratio is not. The CSS30 spec implies that calratio should also be applied.

The variable *ituple* in the FORTRAN routines counts from 1 rather 0. Ie, the first tuple in a relation is tuple #1 in Fortran, and tuple #0 in c.

AUTHOR

Danny Harvey

NAME

scvcreate – create one or more Station-Channel Views

SYNOPSIS

include '/sw/loc/include/SCV_f.i'

call scvcreate(*schema*, *ndbs*, *dbnames*, *stachan*, *tstart*, *tend*, *iexist*, *nscvs*, *iscvs*)

character∗(∗)	*schema*
integer	*ndbs*
character∗(∗)	*dbnames*(*ndbs*)
character∗(∗)	*stachan*
character∗(∗)	*tstart*
character∗(∗)	*tend*
integer	*iexist*
integer	*nscvs*
integer	*iscvs*(*nscvs*)

DESCRIPTION

The FORTRAN subroutine **scvcreate()** will create one or more new Station-Channel views from a variety of different external data formats. This is a FORTRAN version of **SCV_create(3)**.

ARGUMENTS

schema (i) A character string that defines the external data format. The following **schema** strings are currently recognized.

ndbs (i) The number of input names describing the data. Currently this must be one.

dbnames (i) An array of character strings with names describing the input data. There should be **ndbs** strings in the array. The nature of these names is format (schema) dependent. For flat file relational databases these names correspond to the database names, which are flat file names without the *.relname* suffix. The current version will only use the first name.

stachan (i) A station-channel sift key. This string specifies the subset of station-channel data streams that will be read from the input external data. If this string is blank, then no station-channel sifting is done. The string format is:

<sta>:<chan>[,<sta2>:<chan2>,[...]]

where *<sta>* is the station string and *<chan>* is the channel string. The asterisk (∗) character can be used as a wild card in place of *<sta>* or *<chan>* which matches anything. Multiple station-channel specifications are separated by commas (,). Normal UNIX regular expression matching is also used (ala ed). (e.g. A.∗:∗ matches all channels of all stations beginning with A.)

tstart (i) A data sift start time. This string specifies the subset of arrivals and waveform segments that will be read from the input external data. Only arrivals and waveform segments with time later than **tstart** will be read. If this string is blank, then no start time sifting is done. The start time can be specified in several different ways.
1. A character string epoch time in seconds.
2. A character string in 'year:month:day:hour:minute:second' form.
3. A character string in 'julianday:hour:minute:second' form.

tend (i) A data sift end time. This string specifies the subset of arrivals and waveform segments that will be read from the input external data. Only arrivals and waveform segments with time earlier than **tend** will be read. If this string is blank, then no start time sifting is done. The end time is specified in the same manner as the start time.

iexist (i) Waveform existence flag. If this is set to non-zero, then only station-channels with existing waveforms on disk will be read.

nscvs (i/o) The number of Station-Channel views read from the input external data. The

input value defines the maximum size of the **iscvs** array. The output value is set to the actual number of Station-Channel views read from the input external data. If this is 0, then no station-channel streams met the sift requirements. If this is -1, then an error occured.

iscvs (o) The output array of station-channel view handles. These handles must be used in subsequent SCV library calls.

DIAGNOSTICS

SEE ALSO

introscv(3), SCV_create(3)

NAME

scvget – get one or more Station-Channel View attributes

SYNOPSIS

include '/sw/loc/include/SCV_f.i'

call scvget(*iscv***,** *attr1***,** *value1***, [***attr2***,** *value2***], ...,** *0***)**
integer *iscv*

DESCRIPTION

FORTRAN subroutine **scvget()** will get one or more Station-Channel View attributes. Attributes are specified as attribute-value groupings in a variable argument list ala *SunView* or *XView*. This is a *FORTRAN* callable version of **SCV_get(3)**.

ARGUMENTS

iscv (i) A Station-Channel View handle as returned by **scvcreate(3F)**.

This is followed by any number of attribute-value specifications finally terminated by a *0* argument. Following is a list of the attributes and the value types that will be returned.

Attribute	Value Description
SCV_STATION	The station code. Type: character∗6
SCV_CHAN	The channel code. Type: character∗8
SCV_CHANA	The channel alias code. This is used to define channel codes that are longer and more descriptive than the original data format would allow. Type: character∗8
SCV_DBL	The database object handle. This returns a handle to the database object (see **introcssxx(3)**). Type: integer∗4
SCV_TSTART	The epoch start time. This is the earliest time for all waveform segment samples and arrivals. Type: real∗8
SCV_TEND	The epoch end time. This is the latest time for all waveform segment samples and arrivals. Type: real∗8
SCV_LAT	The station latitude in degrees. Type: real∗4
SCV_LON	The station longitude in degrees. Type: real∗4
SCV_ELEV	The station elevation above sea level in km. Type: real∗4
SCV_DEPTH	The channel depth of emplacement in km relative to the station elevation. Type: real∗4
SCV_DEAST	The station relative east position in km relative to some reference station. Type: real∗4
SCV_DNORTH	The station relative north position in km relative to some reference station. Type: real∗4

SCV_DELEV The station relative elevation in km relative to some reference station.
Type: real∗4

SCV_HANG The channel sensor horizontal angle orientation in degrees. This is measured positive clockwise from north.
Type: real∗4

SCV_VANG The channel sensor vertical angle orientation in degrees. This is measured positive down from horizontal.
Type: real∗4

SCV_NSEGS The total number of waveform segments for this station-channel view.
Type: integer

SCV_NARRS The total number of phase arrivals for this station-channel view.
Type: integer

SEE ALSO

introscv(3), scvcreate(3F), SCV_get (3)

NAME

scvgetarrival – get Station-Channel View arrival attributes

SYNOPSIS

subroutine scvgetarrival(*iscv*, *iarr*, [*arrival-id*, *arrival-value*, ...], *SCV_NULL*)
integer *iscv*
integer *iarr*
integer *arrival-id*
integer *arrival-value*
real*8 *arrival-value*
character*8 *arrival-value*

DESCRIPTION

Scvgetarrival() gets arrival attributes from SCV's as returned by scvcreate.

The possible arrival id's and related values are:

SCV_ARR_TUPLE
This returns an integer corresponding to the tuple index of the specified arrival in the arrival relation.

SCV_ARR_TIME
This returns the real*8 arrival time.

SCV_ARR_PHASE
This returns an 8 character phase code.

DIAGNOSTICS

scvgetarrival: Unknown attribute
An unrecognized key was passed in the argument list.

SEE ALSO

introscv(3), scvcreate(3f)

AUTHOR

Danny Harvey

NAME

scvreadwfs, scvgettrace, scvfreewfs – read or free all waveforms associated with an SCV handle

SYNOPSIS

subroutine scvgettrace(*iscv*, *ts*, *te*, *nsegs*, *nsampmax*, *nsamp*, *t0*, *dt*, *data*)
integer *iscv*;
real*8 *ts*;
real*8 *te*;
integer *nsegs*;
integer *nsampmax*;
integer *nsamp*(*nsegs*);
real*8 *t0*(*nsegs*);
real*8 *dt*(*nsegs*);
real *data*(*nsampmax*,*nsegs*);

subroutine scvreadwfs(*iscv*)
integer *iscv*

subroutine scvfreewfs(*iscv*)
integer *iscv*

DESCRIPTION

scvgettrace() reads the waveform data corresponding to the SCV *iscv* for times between *ts* and *te*. The data is returned in the array *data*, as up to *nsegs* separate ''segments'', corresponding to individual continuous data traces. The parameter *nsegs* initially tells scvgettrace the maximum allowable number of segments to return; the actual number of segments returned is returned in this variable. A maximum of *nsampmax* data points are returned in each segment. The start time and sampling increment of each segment data is returned in *t0* and *dt*.

Scvreadwfs() reads all of the waveform data associated with a single SCV handle. Creation of the SCV by scvcreate does not by default read these waveforms; however, running this procedure (if you have adequate memory) will make later calls to **scvgettrace()** much faster.

Scvfreewfs() frees the memory and waveform data read by **scvreadwfs()**.

FILES

Reads data from the data files specified by the wfdisc relation.

SEE ALSO

introscv(3), scvcreate(3f), scvget(3f)

AUTHOR

Danny Harvey

NAME

scvselect – select scvs satisfying sift keys

SYNOPSIS

subroutine scvselect(*nscvsin***,** *scvsin***,** *nscvsout***,** *scvsout***, [***sift-id***,** *sift-value***, ...],** *SCV_NULL*)
integer *nscvsin*
integer *scvsin*(*nscvsin*)
integer *nscvsout*
integer *scvsout*(*nscvsout*)
integer *sift-id*
character *sift-value*(∗)

DESCRIPTION

Scvselect() takes a list of SCV's as returned by **scvcreate()**, and returns a second list consisting of the members from the first list which satisfy certain ''sift'' keys. The ''sift'' keys are specified as an arbitrary number of *id-value* pairs.

The possible sift-key id's and related values are:

SCV_STA / SCV_CHAN

Station and channel values are comma delimited lists of possible station or channel names. Each such name may be the explicit name (eg, KKL) or a UNIX regular expression (eg, K..) (see **grep(1)**). For example, ''KKL,HSU,L.H'' would be a possible station value, and ''H...,...Z'' would be a possible channel value.

SCV_TSTART / SCV_TEND

Tstart and Tend values are strings which can be converted to an epoch time. The SCV's which are returned in *scvsout* will all span a time range which overlaps the range specified by the Tstart and Tend values. The Tstart and Tend values may take three forms:

epoch time in ascii
eg: 557656335.496
year:month:day:hour:min:sec
eg: 1988:feb:15:22:35:43.5
juliandate:hour:min:sec
eg: 1990078:03:10:15.6

DIAGNOSTICS

scvselect: Unknown attribute
An unrecognized key was passed in the argument list.

scvselect: Unable to convert TSTART
It was not possible to convert the starting time string to an epoch time.

scvselect: Unable to convert TEND
It was not possible to convert the ending time string to an epoch time.

SEE ALSO

time_string2epoch(3), introscv(3), scvcreate(3f)

AUTHOR

Danny Harvey

NAME

response – format of instrument response files

DESCRIPTION

Instruments responses are represented in files following a format suggested by CSS. Currently, all responses are represented as poles and zeros only, and the software to use these files is only implemented for poles and zeros. The format, however, accomodates several other representations.

The response files contain only the shape of the response curve, not the actual calibrated response. Actual numbers must be generated using the CALIB, CALRATIO, and CALPER fields from the wfdisc and sensor tables. The instrument table locates the response file for a particular physical device, and CALIB, CALRATIO, and CALPER together specify the relation between measured output and ground motion at the specified frequency (1/CALPER).

The response file is made up of a series of response groups. The division into groups is arbitrary, but might represent different components of the response from different filters or measured versus theoretical response. In the NRDC files, there is only one response group, which is the poles and zeros. The CSS response file format allows an arbitrary number of response groups, which are to be ‘‘cascaded’’ (multiplied) together to form the total response.

Each response group begins with a single line which specifies:

1) response source: either ‘‘theoretical’’ or ‘‘measured’’
2) sequence number: has no real significance, but must be present
3) description: up to 12 characters characterizing the group, eg ‘‘instrument’’, ‘‘anti-alias’’, ...
4) response type: ‘‘fir’’, ‘‘paz’’, or ‘‘fap’’ (only paz is implemented currently in the JSPC software).
5) author/source of information

Comment lines have a ‘‘#’’ in column one, and may be placed anywhere in the file.

FIR

Finite impulse response groups have lines as follows:

1) a line specifying the input samples/second,
2) a line specifying the number of numerator coefficients, followed by
3) that number of lines with each line specifying a numerator and an error.
4) a line specifying the number of denominator coefficients, followed by
5) that number of lines with each line specifying a denominator and an error.

PAZ

Poles and zeros groups (response type ‘‘paz’’) then have the following lines:

1) contains a normalization factor, which is ignored.
2) a line which specifies the number of poles, followed by
3) that number of lines with each line specifying location of a pole (of order 1) and estimated errors in complex coordinates.
4) a line which specifies the number of zeros, followed by
5) that number of lines with each line specifying location of a zero (of order 1) and estimated errors in complex coordinates.

FAP

Frequency, Amplitude, Phase groups contain the following lines:

1) line specifying the number of FAP triplets, followed by
2) that number of lines, with each line specifying frequency (in degrees), amplitude, phase (in

degrees), amplitude error, phase error.

EXAMPLE

```
#
#  Manufacturer: Teledyne Geotech
#  Model        : 54100
#  Description : 3-Component Borehole Seismometer
#  lo = 0.00 npl = 0 aa = 80.00 npa = 6 fs = 0.20 ds = 0.80
#  lo   = Low cut corner
#  npl  = Low cut no. of poles
#  aa   = High cut anti-alias corner
#  npa  = High cut anti-alias no. of poles
#  fs   = Seismometer natural frequency
#  ds   = Seismometer damping
#  Poles come first followed by zeroes
#  All poles and zeroes in radians/sec
#  Transfer function from ground displacement
#  Gain normalizes response at 62.831853 rad/sec (10.000000 Hz)
#
theoretical  1 complete     paz     UCSD/JSPC
2.5673669e+14
8
-1.005310e+00   7.539822e-01    0.0     0.0
-1.005310e+00   -7.539822e-01   0.0     0.0
-4.855273e+02   1.300966e+02    0.0     0.0
-4.855273e+02   -1.300966e+02   0.0     0.0
-3.554306e+02   3.554306e+02    0.0     0.0
-3.554306e+02   -3.554306e+02   0.0     0.0
-1.300966e+02   4.855273e+02    0.0     0.0
-1.300966e+02   -4.855273e+02   0.0     0.0
3
0.0     0.0     0.0     0.0
0.0     0.0     0.0     0.0
0.0     0.0     0.0     0.0
```

BUGS AND CAVEATS

Only the poles and zeros response group is allowed by the current software.

AUTHOR

Daniel Quinlan