NAME

vnadata_alloc, vnadata_init, vnadata_alloc_and_init, vnadata_resize, vnadata_get_type, vnadata_get_type_name, vnadata_set_type, vnadata_get_rows, vnadata_get_columns, vnadata_get_frequencies, vnadata_free, vnadata_get_fmin, vnadata_get_fmax, vnadata_get_frequency, vnadata_set_frequency, vnadata_get_frequency_vector, vnadata_set_frequency_vector, vnadata_add_frequency, vnadata_get_cell, vnadata_set_cell, vnadata_get_matrix, vnadata_set_matrix, vnadata_get_to_vector, vnadata_set_from_vector, vnadata_get_z0, vnadata_set_z0, vnadata_get_z0_vector, vnadata_set_z0_vector, vnadata_set_all_z0, vnadata_has_fz0, vnadata_get_fz0, vnadata_set_fz0, vnadata_get_fz0_vector, vnadata_set_fz0_vector, vnadata_convert, vnadata_load, vnadata_fload, vnadata_save, vnadata_fsave, vnadata_cksave, vnadata_get_filetype, vnadata_set_filetype, vnadata_get_format, vnadata_set_format, vnadata_get_fprecision, vnadata_set_fprecision, vnadata_get_dprecision, vnadata_set_dprecision − Network Parameter Data

SYNOPSIS

#include <vnadata.h>

Link with *-lvna -lm*.

Allocation and Initialization

vnadata_t *vnadata_alloc(vnaerr_error_fn_t *_error_fn_, void *_error_arg_);

int vnadata_init(vnadata_t *_vdp_, vnadata_parameter_type_t _type_, int _rows_, int _columns_, int _frequencies_);

vnadata_t *vnadata_alloc_and_init(vnaerr_error_fn_t *_error_fn_, void *_error_arg_, vnadata_parameter_type_t _type_, int _rows_, int _columns_, int _frequencies_);

int vnadata_resize(vnadata_t *_vdp_, vnadata_parameter_type_t _type_, int _rows_, int _columns_, int _frequencies_);

vnadata_parameter_type_t vnadata_get_type(const vnadata_t *_vdp_);

const char *vnadata_get_type_name(vnadata_parameter_type_t _type_);

int vnadata_set_type(const vnadata_t *_vdp_, vnadata_parameter_type_t _type_);

int vnadata_get_rows(const vnadata_t *_vdp_);

int vnadata_get_columns(const vnadata_t *_vdp_);

int vnadata_get_frequencies(const vnadata_t *_vdp_);

void vnadata_free(vnadata_t *_vdp_);

The Frequency Vector

double vnadata_get_fmin(const vnadata_t *_vdp_);

double vnadata_get_fmax(const vnadata_t *_vdp_);

double vnadata_get_frequency(const vnadata_t *_vdp_, int _findex_);

int vnadata_set_frequency(vnadata_t *_vdp_, int _findex_, double _frequency_);

const double *vnadata_get_frequency_vector(const vnadata_t *_vdp_);

int vnadata_set_frequency_vector(vnadata_t *_vdp_, const double *_frequency_vector_);

int vnadata_add_frequency(vnadata_t *_vdp_, double _frequency_);

Data Elements

double complex vnadata_get_cell(vnadata_t *_vdp_, int _findex_, int _row_, int _column_);

int vnadata_set_cell(vnadata_t *_vdp_, int _findex_, int _row_, int _column_, double complex _value_);

double complex *vnadata_get_matrix(const vnadata_t *_vdp_, int _findex_);

int vnadata_set_matrix(vnadata_t *_vdp_, int _findex_, const double complex *_matrix_);

int vnadata_get_to_vector(const vnadata_t *_vdp_, int _row_, int _column_, double complex *_vector_);

int vnadata_set_from_vector(vnadata_t *_vdp_, int _row_, int _column_, const double complex *_vector_);

Ordinary System Impedances

double complex vnadata_get_z0(const vnadata_t *_vdp_, int _port_);

int vnadata_set_z0(vnadata_t *_vdp_, int _port_, double complex _z0_);

const double complex *vnadata_get_z0_vector(const vnadata_t *_vdp_);

int vnadata_set_z0_vector(vnadata_t *_vdp_, const double complex *_z0_vector_);

int vnadata_set_all_z0(vnadata_t *_vdp_, double complex _z0_);

Frequency-Dependent System Impedances

bool vnadata_has_fz0(const vnadata_t *_vdp_);

double complex vnadata_get_fz0(const vnadata_t *_vdp_, int _findex_, int _port_);

int vnadata_set_fz0(vnadata_t *_vdp_, int _findex_, int _port_, double complex _z0_);

const double complex *vnadata_get_fz0_vector(const vnadata_t *_vdp_, int _findex_);

int vnadata_set_fz0_vector(vnadata_t *_vdp_, int _findex_, const double complex *_z0_vector_);

Parameter Conversion

int vnadata_convert(const vnadata_t *_vdp_in_, vnadata_t *_vdp_out_,
vnadata_parameter_type_t _new_type_);

Load and Save

int vnadata_load(vnadata_t *_vdp_, const char *_filename_);

int vnadata_fload(vnadata_t *_vdp_, FILE *_fp_, const char *_filename_);

int vnadata_save(vnadata_t *_vdp_, const char *_filename_);

int vnadata_fsave(vnadata_t *_vdp_, FILE *_fp_, const char *_filename_);

int vnadata_cksave(vnadata_t *_vdp_, const char *_filename_);

vnadata_filetype_t vnadata_get_filetype(const vnadata_t *_vdp_);

int vnadata_set_filetype(vnadata_t *_vdp_, vnadata_filetype_t _type_);

const char *vnadata_get_format(const vnadata_t *_vdp_);

int vnadata_set_format(vnadata_t *_vdp_, const char *_format_);

int vnadata_get_fprecision(const vnadata_t *_vdp_);

int vnadata_set_fprecision(const vnadata_t *_vdp_, int _fprecision_);

int vnadata_get_dprecision(const vnadata_t *_vdp_);

int vnadata_set_dprecision(const vnadata_t *_vdp_, int _dprecision_);

DESCRIPTION

These functions store and manage electrical network parameter data. Internally, the data are stored as a vector of matrices, one per frequency. The matrix may contain any of s (scattering), t (scattering transfer), u (inverse scattering transfer) z (impedance), y (admittance), h (hybrid), g (inverse-hybrid), a (ABCD), b (inverse ABCD) or zin (input impedance) parameters.

Allocation and Initialization

The **vnadata_alloc**() function allocates an empty **vnadata_t** structure of parameter type VPT_UNDEF. This is useful for creating the empty output container for other functions such as **vnadata_convert**(), or **vnacal_apply**(3).

The **vnadata_init**() function sets the dimensions and parameter type of the **vnadata_t** structure, initializes all frequency and data cells to zero, and initializes all z0 entries to the default of 50 ohms. The *type* argument must be one of: VPT_UNDEF, VPT_S, VPT_T, VPT_U, VPT_Z, VPT_Y, VPT_H, VPT_G, VPT_A, VPT_B, or VPT_ZIN, and the dimensions must be consistent with the parameter type.

The **vnadata_alloc_and_init**() function is a combination of **vnadata_alloc**() and **vnadata_init**().

The **vnadata_resize**() function changes the parameter type and dimensions of the matrix without clearing or converting data. Existing values remain undisturbed when the matrix type, the number of rows, or the number of frequencies are changed, but shift to other cells when the number of columns is changed, as **vnadata_resize**() doesn't reform the matrix. Changing the parameter type with this function doesn't convert existing data to the new type. For type conversion, see **vnadata_convert**().

The **vnadata_get_type**() function returns the current parameter type of the matrix; **vnadata_get_typename**() returns *type* as a string. The **vnadata_set_type**() function changes the parameter type without converting existing data. The *type* parameter must be consistent with the matrix dimensions.

The **vnadata_get_rows**(), **vnadata_get_columns**(), and **vnadata_get_frequencies**() functions return the current dimensions of the matrix.

The **vnadata_free**() function frees the structure and its contents.

The Frequency Vector

The **vnadata_get_fmin**() and **vnadata_get_fmax**() functions return the lowest and highest frequencies, respectively.

The **vnadata_get_frequency**() and **vnadata_set_frequency**() functions, respectively, get and set the frequency at index *findex*.

The **vnadata_get_frequency_vector**() and **vnadata_set_frequency_vector**() functions get and set the entire frequency vector. The length of *frequency_vector* must match *frequencies*.

The **vnadata_add_frequency**() function adds a new frequency entry at the end, filling the associated new data elements with initial values. This function is useful, for example, when parsing a Touchstone V1 file, where you don't know the number of frequencies up-front.

Data Elements

The **vnadata_get_cell**() and **vnadata_set_cell**() functions get and set indivdual data elements. The **vnadata_get_matrix**() and **vnadata_set_matrix**() functions get and set the parameter data matrix for the given frequency. The *matrix* parameter is a pointer to a vector of double complex containing the flattened matrix elements in row-major order.

The **vnadata_set_from_vector**() and **vnadata_get_to_vector**() functions copy a vector of data values, one entry per frequency, into a **vnadata_t** matrix cell, and vice versa. The *vector* argument must point to a vector with length at least the number of frequencies in the **vnadata_t** structure. These functions are useful for translating between the matrix of vectors form used for VNA measurements, and the vector of matrices form used internally by vnadata.

Ordinary System Impedances

The **vnadata_get_z0**() and **vnadata_set_z0**() functions get and set the system impedance for the given *port*. The **vnadata_get_z0_vector**() and **vnadata_set_z0_vector**() functions get and set system impedances for all ports, where the length of *z0_vector* is the maximum of *rows* and *columns*. The **vnadata_set_all_z0**() function sets the system impedances of all ports to the same value, *z0*. If not set, all system impedances default to 50 ohms.

Calling **vnadata_set_z0**(), **vnadata_set_z0_vector**(), or **vnadata_set_all_z0**() when frequency-dependent impedances are in-use (see below) discards all frequency-dependent z0 values and returns to ordinary system impedances with all other impedance values initialized to 50 ohms. If frequency-dependent impedances are in-use, **vnadata_get_z0**() and **vnadata_get_z0_vector**() return failure.

Frequency-Dependent System Impedances

The **vnadata_get_fz0**() and **vnadata_set_fz0**() functions get and set the system impedance for the given port on a per-frequency basis. The **vnadata_get_fz0_vector**() and **vnadata_set_fz0_vector**() functions get and set the system impedances for all ports at a given frequency index, where the length of *z0_vector* is the maximum of *rows* and *columns*.

The **vnadata_has_fz0**() function tests if per-frequency system impedances are in effect and returns true if

they are. If frequency-dependent impedances are not in-use, the **vnadata_set_fz0**() and **vnadata_set_fz0_vector**() functions establish frequency-dependent system impedances, preserving the ordinary system impedances for all other entries. **vnadata_get_fz0**() and **vnadata_get_fz0_vector**() functions work regardless of whether frequency-dependent system impedances are in-effect; in the later case, they don't use the *findex* argument.

Parameter Conversion

The **vnadata_convert**() function converts from one parameter type to another, writing the result into *vdp_out*. If *vdp_out* refers to the same structure as *vdp_in*, then an in-place conversion is done. If *vdp_out* is not the same as *vdp_in* and *new_parameter* is the same type as the input matrix, the data are simply copied. **vnadata_convert**() supports all 72 parameter conversions plus 9 conversions from parameter data to input impedances at each port.

Load and Save

The **vnadata_load**() function loads network parameter data from *filename* into the **vnadata_t** structure, changing the type, dimensions, frequency vector and z0 values of the structure to match the data. If *filename* ends with .ts, .s1p, .s2p, .s3p or .s4p, then **vnadata_load**() loads Touchstone format with version determined from the contents of the file. If *filename* ends in .npd, then **vnadata_load**() loads NPD format. If the type cannot be determined from *filename*, and the **vnadata_t** structure already has a filetype set through **vnadata_set_filetype**() or a previous load, it uses the existing file type. If **vnadata_load**() cannot determine the file type from *filename* or from the **vnadata_t** structure, it defaults to NPD format.

The **vnadata_fload**() function does the same as **vnadata_load**() except that it reads from the already open file pointer *fp*. The *filename* argument to **vnadata_fload**() is used only in error messages, and for determining the file type – it doesn't have to refer to an actual file.

The **vnadata_save**() and **vnadata_fsave**() functions save the contents of the **vnadata_t** structure to *filename* or to the file pointer, *fp*, respectively using the format set by **vnadata_set_format**(). The file type is determined as in **vnadata_load**() with the nuance that Touchstone 1 format can be saved to a file ending in .ts.

vnadata_cksave() checks if we'd be able to save using the current filetype, format and parameter type, without actually doing a save. This function is useful to validate that it will be possible to save with the current save options before doing expensive steps such as measuring data from a device, only to ultimately fail with an error in **vnadata_save**().

The **vnadata_get_file_type**() and **vnadata_set_file_type**() functions get and set the current file type described by the following enumeration:

```
typedef enum vnadata_filetype {
    VNADATA_FILETYPE_AUTO,
    VNADATA_FILETYPE_NPD,
    VNADATA_FILETYPE_TOUCHSTONE1,
    VNADATA_FILETYPE_TOUCHSTONE2
} vnadata_filetype_t;
```

The default is VNAFILE_AUTO which causes the load and save functions to try to determine the file type based on the filename extension. If the file ends with .s<digit>p, the library assumes Touchstone 1 format; if it ends in .ts, the library assumes Touchstone 2 format; if it ends in .npd, the library assumes network parameter data format. When loading Touchstone files, the parser automatically determines the Touchstone version from the contents of the file.

The **vnadata_get_format**() and **vnadata_set_format**() functions get and set the parameter type and units as they appear in the file. The *format* parameter is a comma-separated case-insensitive list of the following specifiers:

S[ri|ma|dB] scattering parameters

T[ri|ma|dB]	scattering-transfer parameters
U[ri|ma|dB]	inverse scattering-transfer parameters
Z[ri|ma]	impedance parameters
Y[ri|ma]	admittance parameters
H[ri|ma]	hybrid parameters
G[ri|ma]	inverse-hybrid parameters
A[ri|ma]	ABCD parameters
B[ri|ma]	inverse ABCD parameters
Zin[ri|ma]	impedance looking into each port
PRC	Zin as parallel resistance and capacitance
PRL	Zin as parallel resistance and inductance
SRC	Zin as series resistance and capacitance
SRL	Zin as series resistance and inducatance
IL	insertion loss (dB)
RL	return loss (dB)
VSWR	voltage standing wave ratio

where the ri, ma or dB suffix is an optional coordinate system modifier:

ri	real, imaginary
ma	magnitude, angle
dB	decibels, angle

In the Touchstone file formats, only one specifier may be given and it must be restricted to one of the s, z, y, h or g variants.

The pointer returned by **vnadata_get_format**() becomes invalid after a call to **vnadata_load**(), **vnadata_fload**() or **vnadata_set_format**().

If **vnadata_set_format**() isn't called, **vnadata_save**() and **vnadata_fsave**() take the parameter type from the **vnadata_t** structure and use default coordinates "ri".

The **vnadata_get_fprecision**(), **vnadata_set_fprecision**(), **vnadata_get_dprecision**(), and **vnadata_set_dprecision**() functions get and set the numeric precision in decimal digits for frequency and data values, respectively, when saving to a file. If not set, *fprecision* defaults to 7 digits and *dprecision* defaults to 6 digits.

RETURN VALUE

On success, the allocate functions return a pointer to a **vnadata_t** structure; the get functions return the requested value, and other integer valued functions return zero. On error, the integer valued functions return -1; the pointer valued functions return NULL; and the double and double complex functions return HUGE_VAL.

ERRORS

See **vnaerr**(3).

EXAMPLES

```
#include <complex.h>
#include <math.h>
#include <stdio.h>
#include <stdlib.h>
#include <vnadata.h>

#define PI      3.14159265
#define FMIN    100e+3          /* Hz */
#define FMAX    1e+9            /* Hz */
#define N       9               /* number of frequencies */
#define L       796e-9          /* Henries */
#define C       318e-12         /* Farads */
```

```
/*
 * error_fn: error printing function for the library
 *   @message: single line error message without a newline
 *   @error_arg: passed through to the error function (unused here)
 *   @category: category of error (ignored here)
 */
static void error_fn(const char *message, void *error_arg,
        vnaerr_category_t category)
{
    (void)fprintf(stderr, "example: %s\n", message);
}

/*
 * main
 */
int main(int argc, char **argv)
{
    vnadata_t *vdp;
    const double fstep = log(FMAX / FMIN) / (double)(N - 1);

    /*
     * Set up Z-parameter matrix for an L-C divider.
     */
    if ((vdp = vnadata_alloc_and_init(error_fn, /*error_arg*/NULL,
                    VPT_Z, 2, 2, N)) == NULL) {
        exit(1);
    }
    for (int findex = 0; findex < N; ++findex) {
        double f = FMIN * exp((double)findex * fstep);
        double complex s = 2 * PI * I * f;
        double complex z[2][2];

        if (vnadata_set_frequency(vdp, findex, f) == -1) {
            exit(2);
        }
        z[0][0] = 1.0 / (C * s) + L * s;
        z[0][1] = 1.0 / (C * s);
        z[1][0] = z[0][1];
        z[1][1] = z[0][1];
        if (vnadata_set_matrix(vdp, findex, &z[0][0]) == -1) {
            exit(3);
        }
    }

    /*
     * Save the parameters in Z real-imaginary, S dB, and Zin
     * magnitude-angle formats.
     */
    if (vnadata_set_format(vdp, "Zri,SdB,Zinma") == -1) {
        exit(4);
    }
    if (vnadata_save(vdp, "vnadata-example.npd") == -1) {
        exit(5);
    }
```

```
/*
 * Print the Z parameters.
 */
(void)printf("z-parameters (real-imaginary)\n");
(void)printf("-------------------------\n");
for (int findex = 0; findex < N; ++findex) {
    double f = vnadata_get_frequency(vdp, findex);

    (void)printf("f %7.2f MHz\n", f / 1.0e+6);
    for (int row = 0; row < 2; ++row) {
        for (int column = 0; column < 2; ++column) {
            double complex value;

            value = vnadata_get_cell(vdp, findex, row, column);
            (void)printf("  %6.1f %6.1f%s",
                    creal(value), cimag(value),
                    column < 1 ? "," : "");
        }
        (void)printf("\n");
    }
    (void)printf("\n");
}
(void)printf("\n");


/*
 * Convert to S-parameters and print.
 */
if (vnadata_convert(vdp, vdp, VPT_S) == -1) {
    exit(6);
}
(void)printf("s-parameters (dB-degrees)\n");
(void)printf("-------------------------\n");
for (int findex = 0; findex < N; ++findex) {
    double f = vnadata_get_frequency(vdp, findex);

    (void)printf("f %7.2f MHz\n", f / 1.0e+6);
    for (int row = 0; row < 2; ++row) {
        for (int column = 0; column < 2; ++column) {
            double complex value;

            value = vnadata_get_cell(vdp, findex, row, column);
            (void)printf("  %5.1f %6.1f%s",
                    20 * log10(cabs(value)), 180 / PI * carg(value),
                    column < 1 ? "," : "");
        }
        (void)printf("\n");
    }
    (void)printf("\n");
}
(void)printf("\n");

/*
 * Convert to impedance into each port and print.
```

```
         */
        if (vnadata_convert(vdp, vdp, VPT_ZIN) == -1) {
            exit(7);
        }
        (void)printf("input-impedances (ohms-degrees)\n");
        (void)printf("------------------------------\n");
        for (int findex = 0; findex < N; ++findex) {
            double f = vnadata_get_frequency(vdp, findex);

            (void)printf("f %7.2f MHz\n", f / 1.0e+6);
            for (int port = 0; port < 2; ++port) {
                double complex value;

                value = vnadata_get_cell(vdp, findex, 0, port);
                (void)printf("  %9.2f %6.1f%s",
                        cabs(value), 180 / PI * carg(value),
                        port < 1 ? "," : "");
            }
            (void)printf("\n");
        }
        (void)printf("\n");
        exit(0);
    }
```

SEE ALSO

vnacal(3), **vnacal_new**(3), **vnaconv**(3), **vnaerr**(3), **vnacal_parameter**(3)