NAME

vnacal_apply, vnacal_apply_m, vnacal_create, vnacal_add_calibration, vnacal_delete_calibration, vnacal_find_calibration, vnacal_free, vnacal_get_calibration_end, vnacal_get_columns, vnacal_get_filename, vnacal_get_fmax, vnacal_get_fmin, vnacal_get_frequencies, vnacal_get_frequency_vector, vnacal_get_name, vnacal_get_rows, vnacal_get_type, vnacal_get_z0, vnacal_load, vnacal_name_to_type, vnacal_property_count, vnacal_property_delete, vnacal_property_get, vnacal_property_get_subtree, vnacal_property_keys, vnacal_property_set, vnacal_property_set_subtree, vnacal_property_type, vnacal_save, vnacal_set_dprecision, vnacal_set_fprecision, vnacal_type_to_name − vector network analyzer calibration

SYNOPSIS

#include <vnacal.h>

Managing and Querying Calibrations

vnacal_t *vnacal_create(vnaerr_error_fn_t *_error_fn_, void *_error_arg_);

vnacal_t *vnacal_load(const char *_pathname_, vnaerr_error_fn_t *_error_fn_, void *_error_arg_);

int vnacal_save(vnacal_t *_vcp_, const char *_pathname_);

int vnacal_add_calibration(vnacal_t *_vcp_, const char *_name_, vnacal_new_t *_vnp_);

int vnacal_find_calibration(const vnacal_t *_vcp_, const char *_name_);

int vnacal_delete_calibration(vnacal_t *_vcp_, int _ci_);

int vnacal_get_calibration_end(const vnacal_t *_vcp_);

const char *vnacal_get_name(const vnacal_t *_vcp_, int _ci_);

vnacal_type_t vnacal_get_type(const vnacal_t *_vcp_, int _ci_);

int vnacal_get_rows(const vnacal_t *_vcp_, int _ci_);

int vnacal_get_columns(const vnacal_t *_vcp_, int _ci_);

int vnacal_get_frequencies(const vnacal_t *_vcp_, int _ci_);

double vnacal_get_fmin(const vnacal_t *_vcp_, int _ci_);

double vnacal_get_fmax(const vnacal_t *_vcp_, int _ci_);

const double *vnacal_get_frequency_vector(const vnacal_t *_vcp_, int _ci_);

double complex vnacal_get_z0(const vnacal_t *_vcp_, int _ci_);

const char *vnacal_get_filename(const vnacal_t *_vcp_);

int vnacal_set_fprecision(vnacal_t *_vcp_, int _precision_);

int vnacal_set_dprecision(vnacal_t *_vcp_, int _precision_);

void vnacal_free(vnacal_t *_vcp_);

vnacal_type_t vnacal_name_to_type(const char *_name_);

const char *vnacal_type_to_name(vnacal_type_t _type_);

Applying a Calibration to Measured Data

int vnacal_apply(vnacal_t *_vcp_, int _ci_, const double *_frequency_vector_, int _frequencies_,
double complex *const *_a_, int _a_rows_, int _a_columns_,
double complex *const *_b_, int _b_rows_, int _b_columns_,
vnadata_t *_s_parameters_);

int vnacal_apply_m(vnacal_t *_vcp_, int _ci_, const double *_frequency_vector_, int _frequencies_,
double complex *const *_m_, int _m_rows_, int _m_columns_,
vnadata_t *_s_parameters_);

Managing User-Defined Properties

int vnacal_property_set(vnacal_t *_vcp_, int _ci_, const char *_format_, ...);

const char *vnacal_property_get(vnacal_t *_vcp_, int _ci_, const char *_format_, ...);

int vnacal_property_delete(vnacal_t *_vcp_, int _ci_, const char *_format_, ...);

int vnacal_property_type(vnacal_t *_vcp_, int _ci_, const char *_format_, ...);

int vnacal_property_count(vnacal_t *_vcp_, int _ci_, const char *_format_, ...);

const char **vnacal_property_keys(vnacal_t *_vcp_, int _ci_, const char *_format_, ...);

vnaproperty_t *vnacal_property_get_subtree(vnacal_t *_vcp_, int _ci_, const char *_format_, ...);

vnaproperty_t **vnacal_property_set_subtree(vnacal_t *_vcp_, int _ci_, const char *_format_, ...);

Link with *-lvna -lyaml -lm*.

DESCRIPTION

These functions calculate error terms for vector network analyzers (VNAs) and use the calculated error terms to convert measured voltages from a device under test (DUT) to an s-parameter matrix.

The library is used in two phases: calibration and application. Calibration begins with a call to **vnacal_create**() to create a **vnacal_t** structure needed by most other vnacal functions. Next, the functions documented in **vnacal_new**(3) beginning with **vnacal_new_alloc**() and ending with **vnacal_new_solve**() build a new calibration and solve for its error correction terms from measurements of calibration standards. The **vnacal_add_calibration**() function stores the new calibration into the **vnacal_t** structure, and finally, **vnacal_save**() saves the new calibraton to a file.

In the application phase, **vnacal_load**() loads the calibration file saved above, and **vnacal_apply**() or **vnacal_apply_m**() apply the calibration to measurements of an unknown device under test to produce the corrected s-parameter matrix. The following sections document these functions in more detail.

Managing and Querying Calibrations

vnacal_create() and **vnacal_load**() return pointers to opaque **vnacal_t** structures used by the other vnacal library functions. **vnacal_create**() is used when we're creating a new calibration file; **vnacal_load**() is used to load previously saved calibration data from a file. The **vnacal_t** structure is a container that stores one or more calibrations, typically calibrations for the same instrument covering different frequency ranges or test configurations. The optional *error_fn* is a pointer to a function the library calls with a single line message (without newline) to report errors; *error_arg* is arbitrary user data passed through to the error function. Both can be NULL. See **vnaerr**(3). Whether an *error_fn* is provided or not, the library functions that can fail set **errno** and return -1 or NULL on failure.

vnacal_save() saves the calibrations stored in the **vnacal_t** structure to the file with name *pathname*.

vnacal_add_calibration() adds a new calibration to the **vnacal_t** structure and returns a calibration index (*ci*) referring to the new calibration, or -1 on error. The *name* argument is a name for the new calibration. If *name* matches an existing calibration in the **vnacal_t** structure, **vnacal_add_calibration**() deletes and replaces the existing calibration. The *vnp* argument is a pointer to a **vnacal_new_t** structure. See **vnacal_new**(3).

vnacal_find_calibration() finds a calibration by name and returns the calibration index, *ci*, or -1 if not found.

vnacal_delete_calibration() deletes the calibration with index *ci* from the **vnacal_t** structure.

vnacal_get_calibration_end() returns one past the highest calibration index which is zero if the **vnacal_t** structure has none. This function can be used as follows to loop through all calibrations:

```
for (int ci = 0; ci < vnacal_get_calibration_end(vcp); ++ci) {
  const char *name;

  if ((name = vnacal_get_name(vcp, ci)) != NULL) { /* skip deleted */
```

```
      printf("%d %s0, ci, name);
    }
  }
```

vnacal_get_name() returns the name of the calibration with calibration index *ci*, or NULL if no calibration has index *ci*.

vnacal_get_type() returns the type of error terms used in the calibration. Refer to **vnacal_new**(3) for the list of types.

vnacal_get_rows() and **vnacal_get_columns**() return the dimensions of the calibration. See **vnacal_new**(3).

vnacal_get_frequencies() returns the number of frequency points used in the calibration

vnacal_get_fmin() and **vnacal_get_fmax**() return the minimum and maximum frequency values, respectively, and **vnacal_get_frequency_vector**() returns a pointer to the full vector of calibration frequencies.

vnacal_get_z0() returns the reference impedance for the given calibration.

vnacal_get_filename() returns a pointer to the file name of the calibration file last loaded from or saved to, or NULL if the **vnacal_t** structure came from **vnacal_create**, and **vnacal_save**() hasn't been called.

vnacal_set_fprecision() and **vnacal_set_dprecision**() set the number of significant figures of precision **vnacal_save**() uses to print frequency and error parameter values, respectively, in the save file. If set to VNACAL_MAX_PRECISION, the library uses hexadecimal floating point notation with no loss of precision. By default, the default frequency precision is 7 and default data precision is 6.

vnacal_free() frees the memory used by the **vnacal_t** structure and any associated **vnacal_new_t** structures.

vnacal_name_to_type() takes an error term type name such as "T8", "U8", "E12", etc. and converts to the corresponding **vnacal_type_t** enum. The name match is case-insensitive. If the name doesn't match any type, the function returns -1. The **vnacal_type_to_name**() function does the opposite: it returns the canonical (upper-case) name for the given *type*.

Applying a Calibration to Measured Data

vnacal_apply() and **vnacal_apply_m**() apply the calibration with index *ci* to measured data and store the resulting s-parameters into the caller provided **vnacal_data_t** structure. The *frequency_vector* argument is a vector of length *frequencies* of frequency points at which the measurements were taken. The range of frequencies must lie within the frequency range of the calibration; however, the frequency points don't have to align with those used during calibration: the library uses rational function interpolation when necessary to interpolate between frequency points.

If the vector network analyzer measures both signal leaving each port (*a* matrix) and signal entering each port (*b* matrix), use **vnacal_apply**(). If it measures only the amount of detected signal, use **vnacal_apply_m**(). In either case, the measurement matrix (*b* or *m*) must be square since each s-parameter, in general, depends on all cells of the measurement matrix. The dimensions of the calibration must also be square, and the same as the measurement matrix, with the exception that a 1x2 or 2x1 calibration can be used with a 2x2 measurement matrix.

For T8, U8, TE10, UE10, T16 and U16 error term types, the *a* matrix has dimensions *b_columns* x *b_columns*. The rows of *a* represent the amount of signal leaving each VNA port; the columns of *a* represent the VNA port that was nominally driving signal when the measurement was taken. When *a* and *b* matrices are given, the library calculates the measurement matrix using ab^{-1}.

For E12 type error terms, the calibration is a *columns* long sequence of independent *rows* x 1 systems; therefore, *a* is a row of 1x1 matrices, or equivalently a row vector of reference values.

The choice of **vnacal_apply**() vs. **vnacal_apply_m**() should be based on which form was used during calibration.

Managing User-Defined Properties

The library provides functions for storing user-defined structures and arrays with the calibrations. This is useful for describing the vector network analyzer, conditions under which a calibration was made, which detector measures which signal, switch settings needed for each measurement, and other information useful to the VNA device software.

All property functions take similar arguments: *vcp* is a pointer to the **vnacal_t** structure; *ci* is the index of the calibration, or -1 to indicate a global property; *format* is a format string as in **sprintf**(); and **...** is a list of additional arguments as appropriate for *format*. The functions use *format* and the additional arguments to construct a string which the property functions interpret. The string must begin with a descriptor consisting of dot-separated identifiers and square-bracket delimited array indices, giving a path through the property tree.

Some example descriptors are: ".", "abc", "abc.def", "[0]", "names[0]", "names[1]", and ".abc.def[2][0].ghi". See **vnaproperty**(3) for complete documentation.

The **vnacal_property_set**() function places a scalar value into the property tree, creating and replacing objects along the path as needed to make them conform to the descriptor string. Normally, the argument to this function has the form *descriptor*=*value*, where everything to the right of the equal sign is taken literally as the value to be set – the right hand side may contain newlines. See **vnaproperty**(3) for additional documentation. Here are some examples:

vnacal_property_set(vcp, -1, "value1=5");
In the global property space, create a key-value map and set *value1* to 5.

vnacal_property_set(vcp, -1, "value2=%d", j);
In the global property space, create a key-value map and set *value2* to the value in variable *j*.

vnacal_property_set(vcp, 0, "my_value%d=%d", i, j);
In calibration zero, create a key-value map using *i* to complete the name and *j* as the value.

vnacal_property_set(vcp, 0, "description=XYZ VNA\nwith 2ft cables");
In calibration zero, create a key-value map and set *description* to the given text.

vnacal_property_set(vcp, ci, "foo.bar=xyz");
Create a key-value map with member *foo* containing a nested key-value map with *bar* set to "xyz".

```
vnacal_property_set(vcp, ci, "detectorMatrix[0][0]=1");
vnacal_property_set(vcp, ci, "detectorMatrix[0][1]=2");
vnacal_property_set(vcp, ci, "detectorMatrix[1][0]=2");
vnacal_property_set(vcp, ci, "detectorMatrix[1][1]=1");
```
Create a key-value map with a nested set of lists under *detectorMatrix*, forming a 2x2 matrix.

```
vnacal_property_set(vcp, ci, "my_reflect[0].name=short");
vnacal_property_set(vcp, ci, "my_reflect[0].gamma=-1.0");
vnacal_property_set(vcp, ci, "my_reflect[1].name=open");
vnacal_property_set(vcp, ci, "my_reflect[1].gamma=1.0");
vnacal_property_set(vcp, ci, "my_reflect[2].name=load");
vnacal_property_set(vcp, ci, "my_reflect[2].gamma=0.0");
```
Create a key-value map with member *my_reflect* containing a list of three key-value maps with *name* and *gamma* sub-members set as shown.

Calling **vnacal_property_set**() on an existing property changes the property to the new value. If the descriptor contains an element with a conflicting type, **vnacal_property_set**() replaces the conflicting element. For example, if after building *my_reflect* in the previous example, we set "my_reflect=newValue", then *my_reflect* changes from a list to a scalar, deleting all six entries we created above. Similarly, setting the root element, ".=newValue", replaces the entire property tree with a scalar.

The **vnacal_property_get**() function retrieves a scalar value from the property tree. For example, after adding the values in the examples above, vnacal_property_get(vcp, ci, "value1") returns the string "5" and vnacal_property_get(vcp, ci, "my_reflect[1].gamma") returns the string "1.0". If the descriptor doesn't refer to a scalar, **vnacal_property_get**() fails with a return of NULL.

The **vnacal_property_delete**() function deletes a property from the tree. For example vnacal_property_delete(vcp, set, "detectorMatrix") deletes *detectorMatrix* and its descendants; vnacal_property_delete(vcp, ci, ".") deletes all properties.

The **vnacal_property_type**() function returns 'm' if the descriptor refers to a key-value map, 'l' if the descriptor refers to a list, or 's' if the descriptor refers to a scalar. Given the detectorMatrix example above:

vnacal_property_type(vcp, ci, ".") returns 'm',
vnacal_property_type(vcp, ci, "detectorMatrix") returns 'l',
vnacal_property_type(vcp, ci, "detectorMatrix[0]") returns 'l', and
vnacal_property_type(vcp, ci, "detectorMatrix[0][0]") returns 's'.

If a component along the path doesn't exist or isn't the specified type, **vnacal_property_type**() fails with a return of -1.

The **vnacal_property_count**() function returns the number of elements in a specified map or list. If applied to a scalar, it fails with a return of -1.

Given a key-value map, **vnacal_property_keys**() returns a vector of pointers to all the keys in the map. The caller is responsible for freeing the returned vector (but not the strings it points to) by a call to **free**(3). If applied to something other than a map, **vnacal_property_keys**() fails with a return of NULL.

The **vnacal_property_get_subtree**() and **vnacal_property_set_subtree**() functions return pointers that can be manipulated by the functions in **vnaproperty**(3). Complete documentation can be found in that page.

RETURN VALUE

The functions that return int set **errno** and return -1 on error. The functions that return pointer types set **errno** and return NULL on error. The **vnacal_get_type**() function returns one of the error parameter type values documented in **vnacal_new**(3), or -1 cast to **vnacal_type_t** on error.

If a non-NULL *error_fn* was passed to **vnacal_create**() or **vnacal_load**(), the **vnacal_create**(), **vnacal_load**(), **vnacal_save**(), **vnacal_add_calibration**(), **vnacal_apply**() and **vnacal_apply_m**() functions call the provided error function with a single line error message before returning failure. See **vnaerr**(3).

The **vnacal_find_calibration**(), **vnacal_delete_calibration**(), all the **vnacal_get_***() functions and the **vnacal_property_***() functions set **errno** and return -1, NULL or HUGE_VAL on failure, but don't invoke the error function. This makes it possible, for example, to use **vnacal_find_calibration**() to test if a given calibration name is present or **vnacal_get_name**() to test if there's a calibration at a given calibration index, without generating error messages. The caller is responsible for reporting any actual errors from these functions.

ERRORS

The library functions reports the following errors:

EBADMSG
The file given to **vnacal_load**() has a syntax error or is otherwise invalid.

EDOM
The *a* matrix given to **vnacal_apply**() or the system of equations used to solve for the s-parameters is singular.

EINVAL
A library function was given an invalid parameter, a key given to one of the **vnacal_property_***() functions has invalid syntax, or a component of the key has a type that doesn't match the property tree.

ENOENT
A file with *pathname* given to **vnacal_load**() doesn't exist, the *name* argument given to **vnacal_find_calibration**() wasn't found, or a **vnacal_property_***() function was given a key that doesn't exist.

ENOMEM
A **malloc**(3), **calloc**(3) or **realloc**(3) call was unable to allocate memory.

ENOPROTOPT
The format version of the file given to **vnacal_load**() is not supported by the library.

In addition, the library can report any error generated by **fopen**(3), **getchar**(3) or **fprintf**(3).

BUGS

You can create calibrations that cannot be used with **vnacal_apply**() or **vnacal_apply_m**(), e.g. a 3x1 calibration. The **vnacal_map_apply**() function needed to use these calibrations hasn't yet been implemented.

EXAMPLES

Example programs can be found in /usr/share/doc/libvna*/examples/ or /usr/local/share/doc/libvna/examples/.

SEE ALSO

vnacal_new(3), **vnaconv**(3), **vnadata**(3), **vnaerr**(3), **vnacal_parameter**(3), **vnaproperty**(3)