A short-hand signature is:

.. parsed-literal::

|FIND\_XXX| (<VAR> name1 [path1 path2 ...])

The general signature is:

.. parsed-literal::

|FIND\_XXX| (

<VAR>

name | |NAMES|

[HINTS [path | ENV var]... ]

[PATHS [path | ENV var]... ]

[REGISTRY\_VIEW (64|32|64\_32|32\_64|HOST|TARGET|BOTH)]

[PATH\_SUFFIXES suffix1 [suffix2 ...]]

[VALIDATOR function]

[DOC "cache documentation string"]

[NO\_CACHE]

[REQUIRED]

[NO\_DEFAULT\_PATH]

[NO\_PACKAGE\_ROOT\_PATH]

[NO\_CMAKE\_PATH]

[NO\_CMAKE\_ENVIRONMENT\_PATH]

[NO\_SYSTEM\_ENVIRONMENT\_PATH]

[NO\_CMAKE\_SYSTEM\_PATH]

[NO\_CMAKE\_INSTALL\_PREFIX]

[CMAKE\_FIND\_ROOT\_PATH\_BOTH |

ONLY\_CMAKE\_FIND\_ROOT\_PATH |

NO\_CMAKE\_FIND\_ROOT\_PATH]

)

This command is used to find a |SEARCH\_XXX\_DESC|.

A cache entry, or a normal variable if ``NO\_CACHE`` is specified,

named by ``<VAR>`` is created to store the result of this command.

If the |SEARCH\_XXX| is found the result is stored in the variable

and the search will not be repeated unless the variable is cleared.

If nothing is found, the result will be ``<VAR>-NOTFOUND``.

Options include:

``NAMES``

Specify one or more possible names for the |SEARCH\_XXX|.

When using this to specify names with and without a version

suffix, we recommend specifying the unversioned name first

so that locally-built packages can be found before those

provided by distributions.

``HINTS``, ``PATHS``

Specify directories to search in addition to the default locations.

The ``ENV var`` sub-option reads paths from a system environment

variable.

.. versionchanged:: 3.24

On ``Windows`` platform, it is possible to include registry queries as part

of the directories, using a :ref:`dedicated syntax <Find Using Windows Registry>`.

Such specifications will be ignored on all other platforms.

``REGISTRY\_VIEW``

.. versionadded:: 3.24

.. include:: FIND\_XXX\_REGISTRY\_VIEW.txt

``PATH\_SUFFIXES``

Specify additional subdirectories to check below each directory

location otherwise considered.

``VALIDATOR``

.. versionadded:: 3.25

Specify a :command:`function` to be called for each candidate item found

(a :command:`macro` cannot be provided, that will result in an error).

Two arguments will be passed to the validator function: the name of a

result variable, and the absolute path to the candidate item. The item

will be accepted and the search will end unless the function sets the

value in the result variable to false in the calling scope. The result

variable will hold a true value when the validator function is entered.

.. parsed-literal::

function(my\_check validator\_result\_var item)

if(NOT item MATCHES ...)

set(${validator\_result\_var} FALSE PARENT\_SCOPE)

endif()

endfunction()

|FIND\_XXX| (result NAMES ... VALIDATOR my\_check)

Note that if a cached result is used, the search is skipped and any

``VALIDATOR`` is ignored. The cached result is not required to pass the

validation function.

``DOC``

Specify the documentation string for the ``<VAR>`` cache entry.

``NO\_CACHE``

.. versionadded:: 3.21

The result of the search will be stored in a normal variable rather than

a cache entry.

.. note::

If the variable is already set before the call (as a normal or cache

variable) then the search will not occur.

.. warning::

This option should be used with caution because it can greatly increase

the cost of repeated configure steps.

``REQUIRED``

.. versionadded:: 3.18

Stop processing with an error message if nothing is found, otherwise

the search will be attempted again the next time |FIND\_XXX| is invoked

with the same variable.

If ``NO\_DEFAULT\_PATH`` is specified, then no additional paths are

added to the search.

If ``NO\_DEFAULT\_PATH`` is not specified, the search process is as follows:

.. |FIND\_PACKAGE\_ROOT\_PREFIX\_PATH\_XXX\_SUBDIR| replace::

|prefix\_XXX\_SUBDIR| for each ``<prefix>`` in the

:variable:`<PackageName>\_ROOT` CMake variable and the

:envvar:`<PackageName>\_ROOT` environment variable if

called from within a find module loaded by

:command:`find\_package(<PackageName>)`

.. |CMAKE\_PREFIX\_PATH\_XXX\_SUBDIR| replace::

|prefix\_XXX\_SUBDIR| for each ``<prefix>`` in :variable:`CMAKE\_PREFIX\_PATH`

.. |ENV\_CMAKE\_PREFIX\_PATH\_XXX\_SUBDIR| replace::

|prefix\_XXX\_SUBDIR| for each ``<prefix>`` in :envvar:`CMAKE\_PREFIX\_PATH`

.. |SYSTEM\_ENVIRONMENT\_PREFIX\_PATH\_XXX\_SUBDIR| replace::

|prefix\_XXX\_SUBDIR| for each ``<prefix>/[s]bin`` in ``PATH``, and

|entry\_XXX\_SUBDIR| for other entries in ``PATH``

.. |CMAKE\_SYSTEM\_PREFIX\_PATH\_XXX\_SUBDIR| replace::

|prefix\_XXX\_SUBDIR| for each ``<prefix>`` in

:variable:`CMAKE\_SYSTEM\_PREFIX\_PATH`

1. If called from within a find module or any other script loaded by a call to

:command:`find\_package(<PackageName>)`, search prefixes unique to the

current package being found. See policy :policy:`CMP0074`.

.. versionadded:: 3.12

Specifically, search paths specified by the following variables, in order:

a. :variable:`<PackageName>\_ROOT` CMake variable,

where ``<PackageName>`` is the case-preserved package name.

b. :variable:`<PACKAGENAME>\_ROOT` CMake variable,

where ``<PACKAGENAME>`` is the upper-cased package name.

See policy :policy:`CMP0144`.

.. versionadded:: 3.27

c. :envvar:`<PackageName>\_ROOT` environment variable,

where ``<PackageName>`` is the case-preserved package name.

d. :envvar:`<PACKAGENAME>\_ROOT` environment variable,

where ``<PACKAGENAME>`` is the upper-cased package name.

See policy :policy:`CMP0144`.

.. versionadded:: 3.27

The package root variables are maintained as a stack, so if called from

nested find modules or config packages, root paths from the parent's find

module or config package will be searched after paths from the current

module or package. In other words, the search order would be

``<CurrentPackage>\_ROOT``, ``ENV{<CurrentPackage>\_ROOT}``,

``<ParentPackage>\_ROOT``, ``ENV{<ParentPackage>\_ROOT}``, etc.

This can be skipped if ``NO\_PACKAGE\_ROOT\_PATH`` is passed or by setting

the :variable:`CMAKE\_FIND\_USE\_PACKAGE\_ROOT\_PATH` to ``FALSE``.

\* |FIND\_PACKAGE\_ROOT\_PREFIX\_PATH\_XXX|

2. Search paths specified in cmake-specific cache variables.

These are intended to be used on the command line with a ``-DVAR=value``.

The values are interpreted as :ref:`semicolon-separated lists <CMake Language Lists>`.

This can be skipped if ``NO\_CMAKE\_PATH`` is passed or by setting the

:variable:`CMAKE\_FIND\_USE\_CMAKE\_PATH` to ``FALSE``.

\* |CMAKE\_PREFIX\_PATH\_XXX|

\* |CMAKE\_XXX\_PATH|

\* |CMAKE\_XXX\_MAC\_PATH|

3. Search paths specified in cmake-specific environment variables.

These are intended to be set in the user's shell configuration,

and therefore use the host's native path separator

(``;`` on Windows and ``:`` on UNIX).

This can be skipped if ``NO\_CMAKE\_ENVIRONMENT\_PATH`` is passed or

by setting the :variable:`CMAKE\_FIND\_USE\_CMAKE\_ENVIRONMENT\_PATH` to ``FALSE``.

\* |ENV\_CMAKE\_PREFIX\_PATH\_XXX|

\* |ENV\_CMAKE\_XXX\_PATH|

\* |ENV\_CMAKE\_XXX\_MAC\_PATH|

4. Search the paths specified by the ``HINTS`` option.

These should be paths computed by system introspection, such as a

hint provided by the location of another item already found.

Hard-coded guesses should be specified with the ``PATHS`` option.

5. Search the standard system environment variables.

This can be skipped if ``NO\_SYSTEM\_ENVIRONMENT\_PATH`` is passed or by

setting the :variable:`CMAKE\_FIND\_USE\_SYSTEM\_ENVIRONMENT\_PATH` to ``FALSE``.

\* |SYSTEM\_ENVIRONMENT\_PATH\_XXX|

|SYSTEM\_ENVIRONMENT\_PATH\_WINDOWS\_XXX|

6. Search cmake variables defined in the Platform files

for the current system. The searching of ``CMAKE\_INSTALL\_PREFIX`` and

``CMAKE\_STAGING\_PREFIX`` can be

skipped if ``NO\_CMAKE\_INSTALL\_PREFIX`` is passed or by setting the

:variable:`CMAKE\_FIND\_USE\_INSTALL\_PREFIX` to ``FALSE``. All these locations

can be skipped if ``NO\_CMAKE\_SYSTEM\_PATH`` is passed or by setting the

:variable:`CMAKE\_FIND\_USE\_CMAKE\_SYSTEM\_PATH` to ``FALSE``.

\* |CMAKE\_SYSTEM\_PREFIX\_PATH\_XXX|

\* |CMAKE\_SYSTEM\_XXX\_PATH|

\* |CMAKE\_SYSTEM\_XXX\_MAC\_PATH|

The platform paths that these variables contain are locations that

typically include installed software. An example being ``/usr/local`` for

UNIX based platforms.

7. Search the paths specified by the PATHS option

or in the short-hand version of the command.

These are typically hard-coded guesses.

The :variable:`CMAKE\_IGNORE\_PATH`, :variable:`CMAKE\_IGNORE\_PREFIX\_PATH`,

:variable:`CMAKE\_SYSTEM\_IGNORE\_PATH` and

:variable:`CMAKE\_SYSTEM\_IGNORE\_PREFIX\_PATH` variables can also cause some

of the above locations to be ignored.

.. versionadded:: 3.16

Added ``CMAKE\_FIND\_USE\_<CATEGORY>\_PATH`` variables to globally disable

various search locations.

.. |FIND\_ARGS\_XXX| replace:: <VAR> NAMES name

On macOS the :variable:`CMAKE\_FIND\_FRAMEWORK` and

:variable:`CMAKE\_FIND\_APPBUNDLE` variables determine the order of

preference between Apple-style and unix-style package components.

.. include:: FIND\_XXX\_ROOT.txt

.. include:: FIND\_XXX\_ORDER.txt