NAME

ctest − CTest Command-Line Reference

Contents

SYNOPSIS

```
ctest [<options>]
ctest --build-and-test <path-to-source> <path-to-build>
      --build-generator <generator> [<options>...]
      [--build-options <opts>...] [--test-command <command> [<args>...]]
ctest {-D <dashboard> | -M <model> -T <action> | -S <script> | -SP <script>}
      [-- <dashboard-options>...]
```

DESCRIPTION

The **ctest** executable is the CMake test driver program. CMake−generated build trees created for projects that use the **enable_testing()** and **add_test()** commands have testing support. This program will run the tests and report results.

OPTIONS

−−preset <preset>, −−preset=<preset>
Use a test preset to specify test options. The project binary directory is inferred from the **configurePreset** key. The current working directory must contain CMake preset files. See **preset** for more details.

−−list−presets
Lists the available test presets. The current working directory must contain CMake preset files.

−C <cfg>, −−build−config <cfg>
Choose configuration to test.

Some CMake−generated build trees can have multiple build configurations in the same tree. This option can be used to specify which one should be tested. Example configurations are **Debug** and **Release**.

−−progress
Enable short progress output from tests.

When the output of **ctest** is being sent directly to a terminal, the progress through the set of tests is reported by updating the same line rather than printing start and end messages for each test on new lines. This can significantly reduce the verbosity of the test output. Test completion messages are still output on their own line for failed tests and the final test summary will also still be logged.

This option can also be enabled by setting the environment variable **CTEST_PROGRESS_OUTPUT**.

−V,−−verbose
Enable verbose output from tests.

Test output is normally suppressed and only summary information is displayed. This option will show all test output.

−VV,−−extra−verbose
Enable more verbose output from tests.

Test output is normally suppressed and only summary information is displayed. This option will show even more test output.

−−debug
Displaying more verbose internals of CTest.

This feature will result in a large number of output that is mostly useful for debugging dashboard problems.

−−output−on−failure
Output anything outputted by the test program if the test should fail. This option can also be enabled by setting the **CTEST_OUTPUT_ON_FAILURE** environment variable

−−stop−on−failure
Stop running the tests when the first failure happens.

−F Enable failover.

This option allows CTest to resume a test set execution that was previously interrupted. If no interruption occurred, the **−F** option will have no effect.

−j <jobs>, −−parallel <jobs>
Run the tests in parallel using the given number of jobs.

This option tells CTest to run the tests in parallel using given number of jobs. This option can also be set by setting the **CTEST_PARALLEL_LEVEL** environment variable.

This option can be used with the **PROCESSORS** test property.

See *Label and Subproject Summary*.

−−resource−spec−file <file>
Run CTest with *resource allocation* enabled, using the *resource specification file* specified in **<file>**.

When **ctest** is run as a *Dashboard Client* this sets the **ResourceSpecFile** option of the *CTest Test Step*.

−−test−load <level>
While running tests in parallel (e.g. with **−j**), try not to start tests when they may cause the CPU load to pass above a given threshold.

When **ctest** is run as a *Dashboard Client* this sets the **TestLoad** option of the *CTest Test Step*.

−Q,−−quiet
Make CTest quiet.

This option will suppress all the output. The output log file will still be generated if the **−−output−log** is specified. Options such as **−−verbose**, **−−extra−verbose**, and **−−debug** are ignored if **−−quiet** is specified.

−O <file>, −−output−log <file>
Output to log file.

This option tells CTest to write all its output to a **<file>** log file.

−−output−junit <file>
Write test results in JUnit format.

This option tells CTest to write test results to **<file>** in JUnit XML format. If **<file>** already exists, it will be overwritten. If using the **−S** option to run a dashboard script, use the **OUTPUT_JUNIT** keyword with the **ctest_test()** command instead.

−N,−−show−only[=<format>]
Disable actual execution of tests.

This option tells CTest to list the tests that would be run but not actually run them. Useful in conjunction with the **−R** and **−E** options.

<format> can be one of the following values.

human Human−friendly output. This is not guaranteed to be stable. This is the default.

json−v1
Dump the test information in JSON format. See *Show as JSON Object Model*.

−L <regex>, −−label−regex <regex>
Run tests with labels matching regular expression as described under string(REGEX).

This option tells CTest to run only the tests whose labels match the given regular expression. When more than one **−L** option is given, a test will only be run if each regular expression matches at least one of the test's labels (i.e. the multiple **−L** labels form an **AND** relationship). See *Label Matching*.

−R <regex>, −−tests−regex <regex>
Run tests matching regular expression.

This option tells CTest to run only the tests whose names match the given regular expression.

−E <regex>, −−exclude−regex <regex>
Exclude tests matching regular expression.

This option tells CTest to NOT run the tests whose names match the given regular expression.

−LE <regex>, −−label−exclude <regex>
Exclude tests with labels matching regular expression.

This option tells CTest to NOT run the tests whose labels match the given regular expression. When more than one **−LE** option is given, a test will only be excluded if each regular expression matches at least one of the test's labels (i.e. the multiple **−LE** labels form an **AND** relationship). See *Label Matching*.

−FA <regex>, −−fixture−exclude−any <regex>
Exclude fixtures matching **<regex>** from automatically adding any tests to the test set.

If a test in the set of tests to be executed requires a particular fixture, that fixture's setup and cleanup tests would normally be added to the test set automatically. This option prevents adding setup or cleanup tests for fixtures matching the **<regex>**. Note that all other fixture behavior is retained, including test dependencies and skipping tests that have fixture setup tests that fail.

−FS <regex>, −−fixture−exclude−setup <regex>
Same as **−FA** except only matching setup tests are excluded.

−FC <regex>, −−fixture−exclude−cleanup <regex>
Same as **−FA** except only matching cleanup tests are excluded.

−D <dashboard>, −−dashboard <dashboard>
Execute dashboard test.

This option tells CTest to act as a CDash client and perform a dashboard test. All tests are **<Mode><Test>**, where **<Mode>** can be **Experimental**, **Nightly**, and **Continuous**, and **<Test>** can be **Start**, **Update**, **Configure**, **Build**, **Test**, **Coverage**, and **Submit**.

See *Dashboard Client*.

−D <var>:<type>=<value>
Define a variable for script mode.

Pass in variable values on the command line. Use in conjunction with **−S** to pass variable values to a dashboard script. Parsing **−D** arguments as variable values is only attempted if the value following **−D** does not match any of the known dashboard types.

−M <model>, −−test−model <model>
Sets the model for a dashboard.

This option tells CTest to act as a CDash client where the **<model>** can be **Experimental**, **Nightly**, and **Continuous**. Combining**−M** and **−T** is similar to **−D**.

See *Dashboard Client*.

−T <action>, −−test−action <action>
Sets the dashboard action to perform.

This option tells CTest to act as a CDash client and perform some action such as **start**, **build**, **test**

etc. See *Dashboard Client Steps* for the full list of actions. Combining **−M** and **−T** is similar to **−D**.

See *Dashboard Client*.

−S <script>, −−script <script>
Execute a dashboard for a configuration.

This option tells CTest to load in a configuration script which sets a number of parameters such as the binary and source directories. Then CTest will do what is required to create and run a dashboard. This option basically sets up a dashboard and then runs**ctest −D** with the appropriate options.

See *Dashboard Client*.

−SP <script>, −−script−new−process <script>
Execute a dashboard for a configuration.

This option does the same operations as **−S** but it will do them in a separate process. This is primarily useful in cases where the script may modify the environment and you do not want the modified environment to impact other **−S** scripts.

See *Dashboard Client*.

−I [Start,End,Stride,test#,test#|Test file], −−tests−information
Run a specific number of tests by number.

This option causes CTest to run tests starting at number **Start**, ending at number **End**, and incrementing by **Stride**. Any additional numbers after **Stride** are considered individual test numbers. **Start**, **End**, or **Stride** can be empty. Optionally a file can be given that contains the same syntax as the command line.

−U, −−union
Take the Union of **−I** and **−R**.

When both **−R** and **−I** are specified by default the intersection of tests are run. By specifying **−U** the union of tests is run instead.

−−rerun−failed
Run only the tests that failed previously.

This option tells CTest to perform only the tests that failed during its previous run. When this option is specified, CTest ignores all other options intended to modify the list of tests to run (**−L**, **−R**, **−E**, **−LE**, **−I**, etc). In the event that CTest runs and no tests fail, subsequent calls to CTest with the **−−rerun−failed** option will run the set of tests that most recently failed (if any).

−−repeat <mode>:<n>
Run tests repeatedly based on the given **<mode>** up to **<n>** times. The modes are:

until−fail
Require each test to run **<n>** times without failing in order to pass. This is useful in finding sporadic failures in test cases.

until−pass
Allow each test to run up to **<n>** times in order to pass. Repeats tests if they fail for any reason. This is useful in tolerating sporadic failures in test cases.

after−timeout
Allow each test to run up to **<n>** times in order to pass. Repeats tests only if they timeout. This is useful in tolerating sporadic timeouts in test cases on busy machines.

−−repeat−until−fail <n>
Equivalent to **−−repeat until−fail:<n>**.

−−max−width <width>
Set the max width for a test name to output.

Set the maximum width for each test name to show in the output. This allows the user to widen the output to avoid clipping the test name which can be very annoying.

−−interactive−debug−mode [0|1]
Set the interactive mode to **0** or **1**.

This option causes CTest to run tests in either an interactive mode or a non−interactive mode. In dashboard mode (**Experimental**, **Nightly**, **Continuous**), the default is non−interactive. In non−interactive mode, the environment variable **DASHBOARD_TEST_FROM_CTEST** is set.

Prior to CMake 3.11, interactive mode on Windows allowed system debug popup windows to appear. Now, due to CTest's use of **libuv** to launch test processes, all system debug popup windows are always blocked.

−−no−label−summary
Disable timing summary information for labels.

This option tells CTest not to print summary information for each label associated with the tests run. If there are no labels on the tests, nothing extra is printed.

See *Label and Subproject Summary*.

−−no−subproject−summary
Disable timing summary information for subprojects.

This option tells CTest not to print summary information for each subproject associated with the tests run. If there are no subprojects on the tests, nothing extra is printed.

See *Label and Subproject Summary*.

−−build−and−test See *Build and Test Mode*.

−−test−dir <dir> Specify the directory in which to look for tests.

−−test−output−size−passed <size>
Limit the output for passed tests to **<size>** bytes.

−−test−output−size−failed <size>
Limit the output for failed tests to **<size>** bytes.

−−overwrite
Overwrite CTest configuration option.

By default CTest uses configuration options from configuration file. This option will overwrite the configuration option.

−−force−new−ctest−process
Run child CTest instances as new processes.

By default CTest will run child CTest instances within the same process. If this behavior is not desired, this argument will enforce new processes for child CTest processes.

−−schedule−random
Use a random order for scheduling tests.

This option will run the tests in a random order. It is commonly used to detect implicit dependencies in a test suite.

−−submit−index
Legacy option for old Dart2 dashboard server feature. Do not use.

−−timeout <seconds>
Set the default test timeout.

This option effectively sets a timeout on all tests that do not already have a timeout set on them via the **TIMEOUT** property.

−−stop−time <time>
Set a time at which all tests should stop running.

Set a real time of day at which all tests should timeout. Example: **7:00:00 −0400**. Any time format understood by the curl date parser is accepted. Local time is assumed if no timezone is specified.

−−print−labels
Print all available test labels.

This option will not run any tests, it will simply print the list of all labels associated with the test set.

−−no−tests=<[error|ignore]>
Regard no tests found either as error or ignore it.

If no tests were found, the default behavior of CTest is to always log an error message but to return an error code in script mode only. This option unifies the behavior of CTest by either returning an error code if no tests were found or by ignoring it.

−−help,−help,−usage,−h,−H,/?
Print usage information and exit.

Usage describes the basic command line interface and its options.

−−version,−version,/V [<f>]
Show program name/version banner and exit.

If a file is specified, the version is written into it. The help is printed to a named <f>ile if given.

−−help−full [<f>]
Print all help manuals and exit.

All manuals are printed in a human−readable text format. The help is printed to a named <f>ile if given.

−−help−manual <man> [<f>]
Print one help manual and exit.

The specified manual is printed in a human−readable text format. The help is printed to a named <f>ile if given.

−−help−manual−list [<f>]
List help manuals available and exit.

The list contains all manuals for which help may be obtained by using the **−−help−manual** option followed by a manual name. The help is printed to a named <f>ile if given.

−−help−command <cmd> [<f>]
Print help for one command and exit.

The **cmake−commands(7)** manual entry for **<cmd>** is printed in a human−readable text format. The help is printed to a named <f>ile if given.

−−help−command−list [<f>]
List commands with help available and exit.

The list contains all commands for which help may be obtained by using the **−−help−command** option followed by a command name. The help is printed to a named <f>ile if given.

−−help−commands [<f>]
Print cmake−commands manual and exit.

The **cmake−commands(7)** manual is printed in a human−readable text format. The help is printed to a named <f>ile if given.

−−help−module <mod> [<f>]
Print help for one module and exit.

The **cmake−modules(7)** manual entry for **<mod>** is printed in a human−readable text format. The help is printed to a named <f>ile if given.

−−help−module−list [<f>]
List modules with help available and exit.

The list contains all modules for which help may be obtained by using the **−−help−module** option followed by a module name. The help is printed to a named <f>ile if given.

−−help−modules [<f>]
Print cmake−modules manual and exit.

The **cmake−modules(7)** manual is printed in a human−readable text format. The help is printed to a named <f>ile if given.

−−help−policy <cmp> [<f>]
Print help for one policy and exit.

The **cmake−policies(7)** manual entry for **<cmp>** is printed in a human−readable text format. The help is printed to a named <f>ile if given.

−−help−policy−list [<f>]
List policies with help available and exit.

The list contains all policies for which help may be obtained by using the **−−help−policy** option followed by a policy name. The help is printed to a named <f>ile if given.

−−help−policies [<f>]
Print cmake−policies manual and exit.

The **cmake−policies(7)** manual is printed in a human−readable text format. The help is printed to a named <f>ile if given.

−−help−property <prop> [<f>]
Print help for one property and exit.

The **cmake−properties(7)** manual entries for **<prop>** are printed in a human−readable text format. The help is printed to a named <f>ile if given.

−−help−property−list [<f>]
List properties with help available and exit.

The list contains all properties for which help may be obtained by using the **−−help−property** option followed by a property name. The help is printed to a named <f>ile if given.

−−help−properties [<f>]
Print cmake−properties manual and exit.

The **cmake−properties(7)** manual is printed in a human−readable text format. The help is printed to a named <f>ile if given.

−−help−variable <var> [<f>]
Print help for one variable and exit.

The **cmake−variables(7)** manual entry for **<var>** is printed in a human−readable text format. The help is printed to a named <f>ile if given.

−−help−variable−list [<f>]
List variables with help available and exit.

The list contains all variables for which help may be obtained by using the **−−help−variable** option followed by a variable name. The help is printed to a named <f>ile if given.

−−help−variables [<f>]
Print cmake−variables manual and exit.

The **cmake−variables(7)** manual is printed in a human−readable text format. The help is printed to a named <f>ile if given.

LABEL MATCHING

Tests may have labels attached to them. Tests may be included or excluded from a test run by filtering on the labels. Each individual filter is a regular expression applied to the labels attached to a test.

When **−L** is used, in order for a test to be included in a test run, each regular expression must match at least one label. Using more than one **−L** option means "match **all** of these".

The **−LE** option works just like **−L**, but excludes tests rather than including them. A test is excluded if each regular expression matches at least one label.

If a test has no labels attached to it, then **−L** will never include that test, and **−LE** will never exclude that test. As an example of tests with labels, consider five tests, with the following labels:

- *test1* has labels *tuesday* and *production*
- *test2* has labels *tuesday* and *test*
- *test3* has labels *wednesday* and *production*
- *test4* has label *wednesday*
- *test5* has labels *friday* and *test*

Running **ctest** with **−L tuesday −L test** will select *test2*, which has both labels. Running CTest with **−L test** will select *test2* and *test5*, because both of them have a label that matches that regular expression.

Because the matching works with regular expressions, take note that running CTest with **−L es** will match all five tests. To select the *tuesday* and *wednesday* tests together, use a single regular expression that

matches either of them, like **−L "tue|wed"**.

LABEL AND SUBPROJECT SUMMARY

CTest prints timing summary information for each **LABEL** and subproject associated with the tests run. The label time summary will not include labels that are mapped to subprojects.

New in version 3.22: Labels added dynamically during test execution are also reported in the timing summary. See Additional Labels.

When the **PROCESSORS** test property is set, CTest will display a weighted test timing result in label and subproject summaries. The time is reported with *sec*proc* instead of just *sec*.

The weighted time summary reported for each label or subproject **j** is computed as:

```
Weighted Time Summary for Label/Subproject j =
    sum(raw_test_time[j,i] * num_processors[j,i], i=1...num_tests[j])

for labels/subprojects j=1...total
```

where:

- **raw_test_time[j,i]**: Wall−clock time for the **i** test for the **j** label or subproject
- **num_processors[j,i]**: Value of the CTest **PROCESSORS** property for the **i** test for the **j** label or subproject
- **num_tests[j]**: Number of tests associated with the **j** label or subproject
- **total**: Total number of labels or subprojects that have at least one test run

Therefore, the weighted time summary for each label or subproject represents the amount of time that CTest gave to run the tests for each label or subproject and gives a good representation of the total expense of the tests for each label or subproject when compared to other labels or subprojects.

For example, if **SubprojectA** showed **100 sec*proc** and **SubprojectB** showed **10 sec*proc**, then CTest allocated approximately 10 times the CPU/core time to run the tests for **SubprojectA** than for **SubprojectB** (e.g. so if effort is going to be expended to reduce the cost of the test suite for the whole project, then reducing the cost of the test suite for **SubprojectA** would likely have a larger impact than effort to reduce the cost of the test suite for **SubprojectB**).

BUILD AND TEST MODE

CTest provides a command−line signature to configure (i.e. run cmake on), build, and/or execute a test:

```
ctest --build-and-test <path-to-source> <path-to-build>
      --build-generator <generator>
      [<options>...]
      [--build-options <opts>...]
      [--test-command <command> [<args>...]]
```

The configure and test steps are optional. The arguments to this command line are the source and binary directories. The**−−b uild−generator** option *must* be provided to use **−−build−and−test**. If**−−test−command** is specified then that will be run after the build is complete. Other options that affect this mode include:

−−build−target

Specify a specific target to build.

If left out the **all** target is built.

−−build−nocmake
Run the build without running cmake first.

Skip the cmake step.

−−build−run−dir
Specify directory to run programs from.

Directory where programs will be after it has been compiled.

−−build−two−config
Run CMake twice.

−−build−exe−dir
Specify the directory for the executable.

−−build−generator
Specify the generator to use. See the **cmake−generators(7)** manual.

−−build−generator−platform
Specify the generator−specific platform.

−−build−generator−toolset
Specify the generator−specific toolset.

−−build−project
Specify the name of the project to build.

−−build−makeprogram
Specify the explicit make program to be used by CMake when configuring and building the project. Only applicable for Make and Ninja based generators.

−−build−noclean
Skip the make clean step.

−−build−config−sample
A sample executable to use to determine the configuration that should be used. e.g. **Deb ug**, **Release** etc.

−−build−options
Additional options for configuring the build (i.e. for CMake, not for the build tool). Note that if this is specified, the **−−build−options** keyword and its arguments must be the last option given on the command line, with the possible exception of **−−test−command**.

−−test−command
The command to run as the test step with the **−−build−and−test** option. All arguments following this keyword will be assumed to be part of the test command line, so it must be the last option given.

−−test−timeout
The time limit in seconds

DASHBOARD CLIENT

CTest can operate as a client for the *CDash* software quality dashboard application. As a dashboard client, CTest performs a sequence of steps to configure, build, and test software, and then submits the results to a *CDash* server. The command−line signature used to submit to *CDash* is:

```
ctest (−D <dashboard> | −M <model> −T <action> | −S <script> | −SP <script>)
      [−− <dashboard−options>...]
```

Options for Dashboard Client include:

−−group <group>
Specify what group you'd like to submit results to

Submit dashboard to specified group instead of default one. By default, the dashboard is submitted to Nightly, Experimental, or Continuous group, but by specifying this option, the group can be arbitrary.

This replaces the deprecated option **−−track**. Despite the name change its behavior is unchanged.

−A <file>, −−add−notes <file>
Add a notes file with submission.

This option tells CTest to include a notes file when submitting dashboard.

−−tomorrow−tag
Nightly or **Experimental** starts with next day tag.

This is useful if the build will not finish in one day.

−−extra−submit <file>[;<file>]
Submit extra files to the dashboard.

This option will submit extra files to the dashboard.

−−http1.0
Submit using *HTTP 1.0*.

This option will force CTest to use *HTTP 1.0* to submit files to the dashboard, instead of *HTTP 1.1*.

−−no−compress−output
Do not compress test output when submitting.

This flag will turn off automatic compression of test output. Use this to maintain compatibility with an older version of CDash which doesn't support compressed test output.

Dashboard Client Steps
CTest defines an ordered list of testing steps of which some or all may be run as a dashboard client:

Start Start a new dashboard submission to be composed of results recorded by the following steps. See the *CTest Start Step* section below.

Update
Update the source tree from its version control repository. Record the old and new versions and the list of updated source files. See the *CTest Update Step* section below.

Configure
Configure the software by running a command in the build tree. Record the configuration output log. See the*CTest Configure Step* section below.

Build Build the software by running a command in the build tree. Record the build output log and detect warnings and errors. See the *CTest Build Step* section below.

Test Test the software by loading a **CTestTestfile.cmake** from the build tree and executing the defined tests. Record the output and result of each test. See the*CTest Test Step* section below.

Coverage
Compute coverage of the source code by running a coverage analysis tool and recording its output. See the *CTest Coverage Step* section below.

MemCheck
Run the software test suite through a memory check tool. Record the test output, results, and issues reported by the tool. See the *CTest MemCheck Step* section below.

Submit
Submit results recorded from other testing steps to the software quality dashboard server. See the *CTest Submit Step* section below.

Dashboard Client Modes
CTest defines three modes of operation as a dashboard client:

Nightly
This mode is intended to be invoked once per day, typically at night. It enables the **Start**, **Update**, **Configure**, **Build**, **Test**, **Coverage**, and **Submit** steps by default. Selected steps run even if the **Update** step reports no changes to the source tree.

Continuous
This mode is intended to be invoked repeatedly throughout the day. It enables the**Start**, **Update**, **Configure**, **Build**, **Test**, **Coverage**, and **Submit** steps by default, but exits after the **Update** step if it reports no changes to the source tree.

Experimental
This mode is intended to be invoked by a developer to test local changes. It enables the **Start**, **Configure**, **Build**, **Test**, **Coverage**, and **Submit** steps by default.

Dashboard Client via CTest Command−Line
CTest can perform testing on an already−generated build tree. Run the **ctest** command with the current working directory set to the build tree and use one of these signatures:

```
ctest −D <mode>[<step>]
ctest −M <mode> [ −T <step> ]...
```

The <**mode**> must be one of the above *Dashboard Client Modes*, and each <**step**> must be one of the above *Dashboard Client Steps*.

CTest reads the *Dashboard Client Configuration* settings from a file in the build tree called either **CTestConfiguration.ini** or **DartConfiguration.tcl** (the names are historical). The format of the file is:

```
# Lines starting in '#' are comments.
# Other non−blank lines are key−value pairs.
<setting>: <value>
```

where <**setting**> is the setting name and <**value**> is the setting value.

In build trees generated by CMake, this configuration file is generated by the **CTest** module if included by the project. The module uses variables to obtain a value for each setting as documented with the settings below.

Dashboard Client via CTest Script
CTest can perform testing driven by a **cmake−language(7)** script that creates and maintains the source and build tree as well as performing the testing steps. Run the **ctest** command with the current working directory set outside of any build tree and use one of these signatures:

```
ctest −S <script>
ctest −SP <script>
```

The <**script**> file must call CTest Commands commands to run testing steps explicitly as documented below. The commands obtain*Dashboar d Client Configuration* settings from their arguments or from variables set in the script.

DASHBOARD CLIENT CONFIGURATION

The *Dashboard Client Steps* may be configured by named settings as documented in the following sections.

CTest Start Step

Start a new dashboard submission to be composed of results recorded by the following steps.

In a *CTest Script*, the **ctest_start()** command runs this step. Arguments to the command may specify some of the step settings. The command first runs the command−line specified by the **CTEST_CHECKOUT_COMMAND** variable, if set, to initialize the source directory.

Configuration settings include:

BuildDirectory
The full path to the project build tree.

- *CTest Script* variable: **CTEST_BINARY_DIRECTORY**
- **CTest** module variable: **PROJECT_BINARY_DIR**

SourceDirectory
The full path to the project source tree.

- *CTest Script* variable: **CTEST_SOURCE_DIRECTORY**
- **CTest** module variable: **PROJECT_SOURCE_DIR**

CTest Update Step

In a *CTest Script*, the **ctest_update()** command runs this step. Arguments to the command may specify some of the step settings.

Configuration settings to specify the version control tool include:

BZRCommand
bzr command−line tool to use if source tree is managed by Bazaar.

- *CTest Script* variable: **CTEST_BZR_COMMAND**
- **CTest** module variable: none

BZRUpdateOptions
Command−line options to the **BZRCommand** when updating the source.

- *CTest Script* variable: **CTEST_BZR_UPDATE_OPTIONS**
- **CTest** module variable: none

CVSCommand
cvs command−line tool to use if source tree is managed by CVS.

- *CTest Script* variable: **CTEST_CVS_COMMAND**
- **CTest** module variable: **CVSCOMMAND**

CVSUpdateOptions
Command−line options to the **CVSCommand** when updating the source.

- *CTest Script* variable: **CTEST_CVS_UPDATE_OPTIONS**
- **CTest** module variable: **CVS_UPDATE_OPTIONS**

GITCommand
git command−line tool to use if source tree is managed by Git.

- *CTest Script* variable: **CTEST_GIT_COMMAND**
- **CTest** module variable: **GITCOMMAND**

The source tree is updated by **git fetch** followed by **git reset −−hard** to the **FETCH_HEAD**. The

result is the same as **git pull** except that any local modifications are overwritten. Use**GITUpdateCustom** to specify a different approach.

GITInitSubmodules
If set, CTest will update the repository's submodules before updating.

- *CTest Script* variable: **CTEST_GIT_INIT_SUBMODULES**
- **CTest** module variable: **CTEST_GIT_INIT_SUBMODULES**

GITUpdateCustom
Specify a custom command line (as a semicolon−separated list) to run in the source tree (Git work tree) to update it instead of running the **GITCommand**.

- *CTest Script* variable: **CTEST_GIT_UPDATE_CUSTOM**
- **CTest** module variable: **CTEST_GIT_UPDATE_CUSTOM**

GITUpdateOptions
Command−line options to the **GITCommand** when updating the source.

- *CTest Script* variable: **CTEST_GIT_UPDATE_OPTIONS**
- **CTest** module variable: **GIT_UPDATE_OPTIONS**

HGCommand
hg command−line tool to use if source tree is managed by Mercurial.

- *CTest Script* variable: **CTEST_HG_COMMAND**
- **CTest** module variable: none

HGUpdateOptions
Command−line options to the **HGCommand** when updating the source.

- *CTest Script* variable: **CTEST_HG_UPDATE_OPTIONS**
- **CTest** module variable: none

P4Client
Value of the **−c** option to the **P4Command**.

- *CTest Script* variable: **CTEST_P4_CLIENT**
- **CTest** module variable: **CTEST_P4_CLIENT**

P4Command
p4 command−line tool to use if source tree is managed by Perforce.

- *CTest Script* variable: **CTEST_P4_COMMAND**
- **CTest** module variable: **P4COMMAND**

P4Options
Command−line options to the **P4Command** for all invocations.

- *CTest Script* variable: **CTEST_P4_OPTIONS**
- **CTest** module variable: **CTEST_P4_OPTIONS**

P4UpdateCustom
Specify a custom command line (as a semicolon−separated list) to run in the source tree (Perforce tree) to update it instead of running the **P4Command**.

- *CTest Script* variable: none
- **CTest** module variable: **CTEST_P4_UPDATE_CUSTOM**

P4UpdateOptions
Command−line options to the **P4Command** when updating the source.

- *CTest Script* variable: **CTEST_P4_UPDATE_OPTIONS**
- **CTest** module variable: **CTEST_P4_UPDATE_OPTIONS**

SVNCommand
svn command−line tool to use if source tree is managed by Subversion.

- *CTest Script* variable: **CTEST_SVN_COMMAND**
- **CTest** module variable: **SVNCOMMAND**

SVNOptions
Command−line options to the **SVNCommand** for all invocations.

- *CTest Script* variable: **CTEST_SVN_OPTIONS**
- **CTest** module variable: **CTEST_SVN_OPTIONS**

SVNUpdateOptions
Command−line options to the **SVNCommand** when updating the source.

- *CTest Script* variable: **CTEST_SVN_UPDATE_OPTIONS**
- **CTest** module variable: **SVN_UPDATE_OPTIONS**

UpdateCommand
Specify the version−control command−line tool to use without detecting the VCS that manages the source tree.

- *CTest Script* variable: **CTEST_UPDATE_COMMAND**
- **CTest** module variable: **<VCS>COMMAND** when **UPDATE_TYPE** is **<vcs>**, else **UPDATE_COMMAND**

UpdateOptions
Command−line options to the **UpdateCommand**.

- *CTest Script* variable: **CTEST_UPDATE_OPTIONS**
- **CTest** module variable: **<VCS>_UPDATE_OPTIONS** when **UPDATE_TYPE** is **<vcs>**, else **UPDATE_OPTIONS**

UpdateType
Specify the version−control system that manages the source tree if it cannot be detected automatically. The value may be **bzr**, **cvs**, **git**, **hg**, **p4**, or **svn**.

- *CTest Script* variable: none, detected from source tree
- **CTest** module variable: **UPDATE_TYPE** if set, else **CTEST_UPDATE_TYPE**

UpdateVersionOnly
Specify that you want the version control update command to only discover the current version that is checked out, and not to update to a different version.

- *CTest Script* variable: **CTEST_UPDATE_VERSION_ONLY**

UpdateVersionOverride
Specify the current version of your source tree.

When this variable is set to a non−empty string, CTest will report the value you specified rather than using the update command to discover the current version that is checked out. Use of this variable supersedes **UpdateVersionOnly**. Like **UpdateVersionOnly**, using this variable tells CTest not to update the source tree to a different version.

- *CTest Script* variable: **CTEST_UPDATE_VERSION_OVERRIDE**

Additional configuration settings include:

NightlyStartTime
In the **Nightly** dashboard mode, specify the "nightly start time". With centralized version control systems (**cvs** and **svn**), the **Update** step checks out the version of the software as of this time so that multiple clients choose a common version to test. This is not well−defined in distributed version−control systems so the setting is ignored.

- *CTest Script* variable: **CTEST_NIGHTLY_START_TIME**
- **CTest** module variable: **NIGHTLY_START_TIME** if set, else **CTEST_NIGHTLY_START_TIME**

CTest Configure Step
In a *CTest Script*, the **ctest_configure()** command runs this step. Arguments to the command may specify some of the step settings.

Configuration settings include:

ConfigureCommand
Command−line to launch the software configuration process. It will be executed in the location specified by the **BuildDirectory** setting.

- *CTest Script* variable: **CTEST_CONFIGURE_COMMAND**
- **CTest** module variable: **CMAKE_COMMAND** followed by **PROJECT_SOURCE_DIR**

LabelsForSubprojects
Specify a semicolon−separated list of labels that will be treated as subprojects. This mapping will be passed on to CDash when configure, test or build results are submitted.

- *CTest Script* variable: **CTEST_LABELS_FOR_SUBPROJECTS**
- **CTest** module variable: **CTEST_LABELS_FOR_SUBPROJECTS**

See *Label and Subproject Summary*.

CTest Build Step
In a *CTest Script*, the **ctest_build()** command runs this step. Arguments to the command may specify some of the step settings.

Configuration settings include:

DefaultCTestConfigurationType
When the build system to be launched allows build−time selection of the configuration (e.g. **Debug**, **Release**), this specifies the default configuration to be built when no **−C** option is given to the **ctest** command. The value will be substituted into the value of **MakeCommand** to replace the literal string **${CTEST_CONFIGURATION_TYPE}** if it appears.

- *CTest Script* variable: **CTEST_CONFIGURATION_TYPE**
- **CTest** module variable: **DEFAULT_CTEST_CONFIGURATION_TYPE**, initialized by the **CMAKE_CONFIG_TYPE** environment variable

LabelsForSubprojects
Specify a semicolon−separated list of labels that will be treated as subprojects. This mapping will be passed on to CDash when configure, test or build results are submitted.

- *CTest Script* variable: **CTEST_LABELS_FOR_SUBPROJECTS**
- **CTest** module variable: **CTEST_LABELS_FOR_SUBPROJECTS**

See *Label and Subproject Summary*.

MakeCommand
Command−line to launch the software build process. It will be executed in the location specified by the **BuildDirectory** setting.

- *CTest Script* variable: **CTEST_BUILD_COMMAND**
- **CTest** module variable: **MAKECOMMAND**, initialized by the **build_command()** command

UseLaunchers
For build trees generated by CMake using one of the Makefile Generators or the **Ninja** generator, specify whether the **CTEST_USE_LAUNCHERS** feature is enabled by the **CTestUseLaunchers** module (also included by the **CTest** module). When enabled, the generated build system wraps each invocation of the compiler, linker, or custom command line with a "launcher" that communicates with CTest via environment variables and files to report granular build warning and error information. Otherwise, CTest must "scrape" the build output log for diagnostics.

- *CTest Script* variable: **CTEST_USE_LAUNCHERS**
- **CTest** module variable: **CTEST_USE_LAUNCHERS**

CTest Test Step
In a *CTest Script*, the **ctest_test()** command runs this step. Arguments to the command may specify some of the step settings.

Configuration settings include:

ResourceSpecFile
Specify a *resource specification file*.

- *CTest Script* variable: **CTEST_RESOURCE_SPEC_FILE**
- **CTest** module variable: **CTEST_RESOURCE_SPEC_FILE**

See *Resource Allocation* for more information.

LabelsForSubprojects
Specify a semicolon−separated list of labels that will be treated as subprojects. This mapping will be passed on to CDash when configure, test or build results are submitted.

- *CTest Script* variable: **CTEST_LABELS_FOR_SUBPROJECTS**
- **CTest** module variable: **CTEST_LABELS_FOR_SUBPROJECTS**

See *Label and Subproject Summary*.

TestLoad
While running tests in parallel (e.g. with **−j**), try not to start tests when they may cause the CPU load to pass above a given threshold.

- *CTest Script* variable: **CTEST_TEST_LOAD**
- **CTest** module variable: **CTEST_TEST_LOAD**

TimeOut
The default timeout for each test if not specified by the **TIMEOUT** test property.

- *CTest Script* variable: **CTEST_TEST_TIMEOUT**
- **CTest** module variable: **DART_TESTING_TIMEOUT**

To report extra test values to CDash, see Additional Test Measurements.

CTest Coverage Step
In a *CTest Script*, the **ctest_coverage()** command runs this step. Arguments to the command may specify some of the step settings.

Configuration settings include:

CoverageCommand
Command−line tool to perform software coverage analysis. It will be executed in the location specified by the **BuildDirectory** setting.

- *CTest Script* variable: **CTEST_COVERAGE_COMMAND**
- **CTest** module variable: **COVERAGE_COMMAND**

CoverageExtraFlags
Specify command−line options to the **CoverageCommand** tool.

- *CTest Script* variable: **CTEST_COVERAGE_EXTRA_FLAGS**
- **CTest** module variable: **COVERAGE_EXTRA_FLAGS**

These options are the first arguments passed to **CoverageCommand**.

CTest MemCheck Step
In a *CTest Script*, the **ctest_memcheck()** command runs this step. Arguments to the command may specify some of the step settings.

Configuration settings include:

MemoryCheckCommand
Command−line tool to perform dynamic analysis. Test command lines will be launched through this tool.

- *CTest Script* variable: **CTEST_MEMORYCHECK_COMMAND**
- **CTest** module variable: **MEMORYCHECK_COMMAND**

MemoryCheckCommandOptions
Specify command−line options to the **MemoryCheckCommand** tool. They will be placed prior to the test command line.

- *CTest Script* variable: **CTEST_MEMORYCHECK_COMMAND_OPTIONS**
- **CTest** module variable: **MEMORYCHECK_COMMAND_OPTIONS**

MemoryCheckType
Specify the type of memory checking to perform.

- *CTest Script* variable: **CTEST_MEMORYCHECK_TYPE**
- **CTest** module variable: **MEMORYCHECK_TYPE**

MemoryCheckSanitizerOptions
Specify options to sanitizers when running with a sanitize−enabled build.

- *CTest Script* variable: **CTEST_MEMORYCHECK_SANITIZER_OPTIONS**
- **CTest** module variable: **MEMORYCHECK_SANITIZER_OPTIONS**

MemoryCheckSuppressionFile
Specify a file containing suppression rules for the **MemoryCheckCommand** tool. It will be passed with options appropriate to the tool.

- *CTest Script* variable: **CTEST_MEMORYCHECK_SUPPRESSIONS_FILE**
- **CTest** module variable: **MEMORYCHECK_SUPPRESSIONS_FILE**

Additional configuration settings include:

BoundsCheckerCommand
Specify a **MemoryCheckCommand** that is known to be command−line compatible with Bounds Checker.

- *CTest Script* variable: none
- **CTest** module variable: none

PurifyCommand
Specify a **MemoryCheckCommand** that is known to be command−line compatible with Purify.

- *CTest Script* variable: none
- **CTest** module variable: **PURIFYCOMMAND**

ValgrindCommand
Specify a **MemoryCheckCommand** that is known to be command−line compatible with Valgrind.

- *CTest Script* variable: none
- **CTest** module variable: **VALGRIND_COMMAND**

ValgrindCommandOptions
Specify command−line options to the **ValgrindCommand** tool. They will be placed prior to the test command line.

- *CTest Script* variable: none
- **CTest** module variable: **VALGRIND_COMMAND_OPTIONS**

DrMemoryCommand
Specify a **MemoryCheckCommand** that is known to be a command−line compatible with DrMemory.

- *CTest Script* variable: none
- **CTest** module variable: **DRMEMORY_COMMAND**

DrMemoryCommandOptions
Specify command−line options to the **DrMemoryCommand** tool. They will be placed prior to the test command line.

- *CTest Script* variable: none
- **CTest** module variable: **DRMEMORY_COMMAND_OPTIONS**

CudaSanitizerCommand
Specify a **MemoryCheckCommand** that is known to be a command−line compatible with cuda−memcheck or compute−sanitizer.

- *CTest Script* variable: none
- **CTest** module variable: **CUDA_SANITIZER_COMMAND**

CudaSanitizerCommandOptions
Specify command−line options to the **CudaSanitizerCommand** tool. They will be placed prior to the test command line.

- *CTest Script* variable: none
- **CTest** module variable: **CUDA_SANITIZER_COMMAND_OPTIONS**

CTest Submit Step
In a *CTest Script*, the **ctest_submit()** command runs this step. Arguments to the command may specify some of the step settings.

Configuration settings include:

BuildName
Describe the dashboard client platform with a short string. (Operating system, compiler, etc.)

- *CTest Script* variable: **CTEST_BUILD_NAME**
- **CTest** module variable: **BUILDNAME**

CDashVersion
Legacy option. Not used.

- *CTest Script* variable: none, detected from server
- **CTest** module variable: **CTEST_CDASH_VERSION**

CTestSubmitRetryCount
Specify a number of attempts to retry submission on network failure.

- *CTest Script* variable: none, use the **ctest_submit() RETRY_COUNT** option.
- **CTest** module variable: **CTEST_SUBMIT_RETRY_COUNT**

CTestSubmitRetryDelay
Specify a delay before retrying submission on network failure.

- *CTest Script* variable: none, use the **ctest_submit() RETRY_DELAY** option.
- **CTest** module variable: **CTEST_SUBMIT_RETRY_DELAY**

CurlOptions
Specify a semicolon−separated list of options to control the Curl library that CTest uses internally to connect to the server. Possible options are**CURLOPT_SSL_VERIFYPEER_OFF** and **CURLOPT_SSL_VERIFYHOST_OFF**.

- *CTest Script* variable: **CTEST_CURL_OPTIONS**
- **CTest** module variable: **CTEST_CURL_OPTIONS**

DropLocation
Legacy option. When **SubmitURL** is not set, it is constructed from **DropMethod**, **DropSiteUser**, **DropSitePassword**, **DropSite**, and **DropLocation**.

- *CTest Script* variable: **CTEST_DROP_LOCATION**
- **CTest** module variable: **DROP_LOCATION** if set, else **CTEST_DROP_LOCATION**

DropMethod
Legacy option. When **SubmitURL** is not set, it is constructed from **DropMethod**, **DropSiteUser**, **DropSitePassword**, **DropSite**, and **DropLocation**.

- *CTest Script* variable: **CTEST_DROP_METHOD**
- **CTest** module variable: **DROP_METHOD** if set, else **CTEST_DROP_METHOD**

DropSite
Legacy option. When **SubmitURL** is not set, it is constructed from **DropMethod**, **DropSiteUser**, **DropSitePassword**, **DropSite**, and **DropLocation**.

- *CTest Script* variable: **CTEST_DROP_SITE**
- **CTest** module variable: **DROP_SITE** if set, else **CTEST_DROP_SITE**

DropSitePassword
Legacy option. When **SubmitURL** is not set, it is constructed from **DropMethod**, **DropSiteUser**, **DropSitePassword**, **DropSite**, and **DropLocation**.

- *CTest Script* variable: **CTEST_DROP_SITE_PASSWORD**
- **CTest** module variable: **DROP_SITE_PASSWORD** if set, else **CTEST_DROP_SITE_PASSWORD**

DropSiteUser
Legacy option. When **SubmitURL** is not set, it is constructed from **DropMethod**, **DropSiteUser**, **DropSitePassword**, **DropSite**, and **DropLocation**.

- *CTest Script* variable: **CTEST_DROP_SITE_USER**
- **CTest** module variable: **DROP_SITE_USER** if set, else **CTEST_DROP_SITE_USER**

IsCDash
Legacy option. Not used.

- *CTest Script* variable: **CTEST_DROP_SITE_CDASH**
- **CTest** module variable: **CTEST_DROP_SITE_CDASH**

ScpCommand
Legacy option. Not used.

- *CTest Script* variable: **CTEST_SCP_COMMAND**
- **CTest** module variable: **SCPCOMMAND**

Site Describe the dashboard client host site with a short string. (Hostname, domain, etc.)

- *CTest Script* variable: **CTEST_SITE**
- **CTest** module variable: **SITE**, initialized by the **site_name()** command

SubmitURL
The **http** or **https** URL of the dashboard server to send the submission to.

- *CTest Script* variable: **CTEST_SUBMIT_URL**
- **CTest** module variable: **SUBMIT_URL** if set, else **CTEST_SUBMIT_URL**

TriggerSite
Legacy option. Not used.

- *CTest Script* variable: **CTEST_TRIGGER_SITE**
- **CTest** module variable: **TRIGGER_SITE** if set, else **CTEST_TRIGGER_SITE**

SHOW AS JSON OBJECT MODEL

When the **−−show−only=json−v1** command line option is given, the test information is output in JSON format. Version 1.0 of the JSON object model is defined as follows:

kind The string "ctestInfo".

version
A JSON object specifying the version components. Its members are

major A non−negative integer specifying the major version component.

minor A non−negative integer specifying the minor version component.

backtraceGraph
JSON object representing backtrace information with the following members:

commands
List of command names.

files List of file names.

nodes List of node JSON objects with members:

command
Index into the **commands** member of the **backtraceGraph**.

file Index into the **files** member of the **backtraceGraph**.

line Line number in the file where the backtrace was added.

parent Index into the **nodes** member of the **backtraceGraph** representing the parent in the graph.

tests A JSON array listing information about each test. Each entry is a JSON object with members:

name Test name.

config Configuration that the test can run on. Empty string means any config.

command
List where the first element is the test command and the remaining elements are the command arguments.

backtrace
Index into the **nodes** member of the **backtraceGraph**.

properties
Test properties. Can contain keys for each of the supported test properties.

RESOURCE ALLOCATION

CTest provides a mechanism for tests to specify the resources that they need in a fine−grained way, and for users to specify the resources available on the running machine. This allows CTest to internally keep track of which resources are in use and which are free, scheduling tests in a way that prevents them from trying to claim resources that are not available.

When the resource allocation feature is used, CTest will not oversubscribe resources. For example, if a resource has 8 slots, CTest will not run tests that collectively use more than 8 slots at a time. This has the effect of limiting how many tests can run at any given time, even if a high **−j** argument is used, if those tests all use some slots from the same resource. In addition, it means that a single test that uses more of a resource than is available on a machine will not run at all (and will be reported as **Not Run**).

A common use case for this feature is for tests that require the use of a GPU. Multiple tests can simultaneously allocate memory from a GPU, but if too many tests try to do this at once, some of them will fail to allocate, resulting in a failed test, even though the test would have succeeded if it had the memory it needed. By using the resource allocation feature, each test can specify how much memory it requires from a GPU, allowing CTest to schedule tests in a way that running several of these tests at once does not exhaust the GPU's memory pool.

Please note that CTest has no concept of what a GPU is or how much memory it has, nor does it have any way of communicating with a GPU to retrieve this information or perform any memory management. CTest simply keeps track of a list of abstract resource types, each of which has a certain number of slots available for tests to use. Each test specifies the number of slots that it requires from a certain resource, and CTest then schedules them in a way that prevents the total number of slots in use from exceeding the listed capacity. When a test is executed, and slots from a resource are allocated to that test, tests may assume that they have exclusive use of those slots for the duration of the test's process.

The CTest resource allocation feature consists of two inputs:

- The *resource specification file*, described below, which describes the resources available on the system.
- The **RESOURCE_GROUPS** property of tests, which describes the resources required by the test.

When CTest runs a test, the resources allocated to that test are passed in the form of a set of *environment variables* as described below. Using this information to decide which resource to connect to is left to the test writer.

The **RESOURCE_GROUPS** property tells CTest what resources a test expects to use grouped in a way meaningful to the test. The test itself must read the *environment variables* to determine which resources have been allocated to each group. For example, each group may correspond to a process the test will spawn when executed.

Note that even if a test specifies a **RESOURCE_GROUPS** property, it is still possible for that to test to run

without any resource allocation (and without the corresponding *environment variables*) if the user does not pass a resource specification file. Passing this file, either through the **--resource-spec-file** command-line argument or the **RESOURCE_SPEC_FILE** argument to **ctest_test()**, is what activates the resource allocation feature. Tests should check the **CTEST_RESOURCE_GROUP_COUNT** environment variable to find out whether or not resource allocation is activated. This variable will always (and only) be defined if resource allocation is activated. If resource allocation is not activated, then the **CTEST_RESOURCE_GROUP_COUNT** variable will not exist, even if it exists for the parent **ctest** process. If a test absolutely must have resource allocation, then it can return a failing exit code or use the **SKIP_RETURN_CODE** or **SKIP_REGULAR_EXPRESSION** properties to indicate a skipped test.

Resource Specification File

The resource specification file is a JSON file which is passed to CTest, either on the *ctest(1)* command line as **--resource-spec-file**, or as the **RESOURCE_SPEC_FILE** argument of **ctest_test()**. If a dashboard script is used and **RESOURCE_SPEC_FILE** is not specified, the value of **CTEST_RESOURCE_SPEC_FILE** in the dashboard script is used instead. If **--resource-spec-file**, **RESOURCE_SPEC_FILE**, and **CTEST_RESOURCE_SPEC_FILE** in the dashboard script are not specified, the value of **CTEST_RESOURCE_SPEC_FILE** in the CMake build is used instead. If none of these are specified, no resource spec file is used.

The resource specification file must be a JSON object. All examples in this document assume the following resource specification file:

```
{
  "version": {
    "major": 1,
    "minor": 0
  },
  "local": [
    {
      "gpus": [
        {
          "id": "0",
          "slots": 2
        },
        {
          "id": "1",
          "slots": 4
        },
        {
          "id": "2",
          "slots": 2
        },
        {
          "id": "3"
        }
      ],
      "crypto_chips": [
        {
          "id": "card0",
          "slots": 4
        }
      ]
    }
  ]
}
```

The members are:

version
An object containing a **major** integer field and a **minor** integer field. Currently, the only supported version is major **1**, minor **0**. Any other value is an error.

local A JSON array of resource sets present on the system. Currently, this array is restricted to being of size 1.

Each array element is a JSON object with members whose names are equal to the desired resource types, such as **gpus**. These names must start with a lowercase letter or an underscore, and subsequent characters can be a lowercase letter, a digit, or an underscore. Uppercase letters are not allowed, because certain platforms have case−insensitive environment variables. See the *Environment Variables* section below for more information. It is recommended that the resource type name be the plural of a noun, such as **gpus** or **crypto_chips** (and not **gpu** or **crypto_chip**.)

Please note that the names **gpus** and **crypto_chips** are just examples, and CTest does not interpret them in any way. You are free to make up any resource type you want to meet your own requirements.

The value for each resource type is a JSON array consisting of JSON objects, each of which describe a specific instance of the specified resource. These objects have the following members:

id A string consisting of an identifier for the resource. Each character in the identifier can be a lowercase letter, a digit, or an underscore. Uppercase letters are not allowed.

Identifiers must be unique within a resource type. However, they do not have to be unique across resource types. For example, it is valid to have a **gpus** resource named **0** and a **crypto_chips** resource named **0**, but not two **gpus** resources both named **0**.

Please note that the IDs **0**, **1**, **2**, **3**, and **card0** are just examples, and CTest does not interpret them in any way. You are free to make up any IDs you want to meet your own requirements.

slots An optional unsigned number specifying the number of slots available on the resource. For example, this could be megabytes of RAM on a GPU, or cryptography units available on a cryptography chip. If **slots** is not specified, a default value of **1** is assumed.

In the example file above, there are four GPUs with ID's 0 through 3. GPU 0 has 2 slots, GPU 1 has 4, GPU 2 has 2, and GPU 3 has a default of 1 slot. There is also one cryptography chip with 4 slots.

RESOURCE_GROUPS Property
See **RESOURCE_GROUPS** for a description of this property.

Environment Variables
Once CTest has decided which resources to allocate to a test, it passes this information to the test executable as a series of environment variables. For each example below, we will assume that the test in question has a **RESOURCE_GROUPS** property of **2,gpus:2;gpus:4,gpus:1,crypto_chips:2**.

The following variables are passed to the test process:

CTEST_RESOURCE_GROUP_COUNT
The total number of groups specified by the **RESOURCE_GROUPS** property. For example:

- **CTEST_RESOURCE_GROUP_COUNT=3**

This variable will only be defined if *ctest(1)* has been given a **−−resource−spec−file**, or if **ctest_test()** has been given a **RESOURCE_SPEC_FILE**. If no resource specification file has been given, this variable will not be defined.

CTEST_RESOURCE_GROUP_<num>
The list of resource types allocated to each group, with each item separated by a comma. **<num>** is a number from zero to **CTEST_RESOURCE_GROUP_COUNT** minus one. **CTEST_RESOURCE_GROUP_<num>** is defined for each **<num>** in this range. For example:

- **CTEST_RESOURCE_GROUP_0=gpus**
- **CTEST_RESOURCE_GROUP_1=gpus**
- **CTEST_RESOURCE_GROUP_2=crypto_chips,gpus**

CTEST_RESOURCE_GROUP_<num>_<resource−type>
The list of resource IDs and number of slots from each ID allocated to each group for a given resource type. This variable consists of a series of pairs, each pair separated by a semicolon, and with the two items in the pair separated by a comma. The first item in each pair is **id:** followed by the ID of a resource of type **<resource−type>**, and the second item is **slots:** followed by the number of slots from that resource allocated to the given group. For example:

- **CTEST_RESOURCE_GROUP_0_GPUS=id:0,slots:2**
- **CTEST_RESOURCE_GROUP_1_GPUS=id:2,slots:2**
- **CTEST_RESOURCE_GROUP_2_GPUS=id:1,slots:4;id:3,slots:1**
- **CTEST_RESOURCE_GROUP_2_CRYPTO_CHIPS=id:card0,slots:2**

In this example, group 0 gets 2 slots from GPU **0**, group 1 gets 2 slots from GPU **2**, and group 2 gets 4 slots from GPU **1**, 1 slot from GPU **3**, and 2 slots from cryptography chip **card0**.

<num> is a number from zero to **CTEST_RESOURCE_GROUP_COUNT** minus one. **<resource−type>** is the name of a resource type, converted to uppercase. **CTEST_RESOURCE_GROUP_<num>_<resource−type>** is defined for the product of each **<num>** in the range listed above and each resource type listed in **CTEST_RESOURCE_GROUP_<num>**.

Because some platforms have case−insensitive names for environment variables, the names of resource types may not clash in a case−insensitive environment. Because of this, for the sake of simplicity, all resource types must be listed in all lowercase in the *resource specification file* and in the **RESOURCE_GROUPS** property, and they are converted to all uppercase in the **CTEST_RESOURCE_GROUP_<num>_<resource−type>** environment variable.

SEE ALSO

The following resources are available to get help using CMake:

Home Page
https://cmake.org

The primary starting point for learning about CMake.

Online Documentation and Community Resources
https://cmake.org/documentation

Links to available documentation and community resources may be found on this web page.

Discourse Forum
https://discourse.cmake.org

The Discourse Forum hosts discussion and questions about CMake.

: *https://cdash.org*

COPYRIGHT