NAME

Kyuafile - Test suite description files

SYNOPSIS

atf_test_program(*string name*, *[string metadata]*);

current_kyuafile();

fs.basename(*string path*);

fs.dirname(*string path*);

fs.exists(*string path*);

fs.files(*string path*);

fs.is_absolute(*string path*);

fs.join(*string path*, *string path*);

include(*string path*);

plain_test_program(*string name*, *[string metadata]*);

syntax(*int version*);

tap_test_program(*string name*, *[string metadata]*);

test_suite(*string name*);

DESCRIPTION

A test suite is a collection of test programs and is represented by a hierarchical layout of test binaries on the file system. Any subtree of the file system can represent a test suite, provided that it includes one or more **Kyuafile**s, which are the test suite definition files.

A **Kyuafile** is a Lua script whose purpose is to describe the structure of the test suite it belongs to. To do so, the script has access to a collection of special functions provided by kyua(1) as described in *Helper functions*.

File versioning

Every **Kyuafile** file starts with a call to **syntax**(*int version*). This call determines the specific schema used by the file so that future backwards-incompatible modifications to the file can be introduced.

Any new **Kyuafile** file should set *version* to ‘2’.

Test suite definition

If the **Kyuafile** registers any test programs, the **Kyuafile** must define the name of the test suite the test programs belong to by using the **test_suite**() function at the very beginning of the file.

The test suite name provided in the **test_suite**() call tells kyua(1) which set of configuration variables from kyua.conf(5) to pass to the test programs at run time.

Test program registration

A **Kyuafile** can register test programs by means of a variety of ***_test_program**() functions, all of which take the name of a test program and a set of optional metadata properties that describe such test program.

The test programs to be registered must live in the current directory; in other words, the various ***_test_program**() calls cannot reference test programs in other directories. The rationale for this is to force all **Kyuafile** files to be self-contained, and to simplify their internal representation.

ATF test programs are those that use the atf(7) libraries. They can be registered with the **atf_test_program**() table constructor. This function takes the *name* of the test program and a collection of optional metadata settings for all the test cases in the test program. Any metadata properties defined by the test cases themselves override the metadata values defined here.

Plain test programs are those that return 0 on success and non-0 on failure; in general, most test programs (even those that use fancy unit-testing libraries) behave this way and thus also qualify as plain test programs. They can be registered with the **plain_test_program**() table constructor. This function takes the *name* of the test program, an optional *test_suite* name that overrides the global test suite name, and a collection of optional metadata settings for the test program.

TAP test programs are those that implement the Test Anything Protocol. They can be registered with the **tap_test_program**() table constructor. This function takes the *name* of the test program and a collection of optional metadata settings for the test program.

The following metadata properties can be passed to any test program definition:

allowed_architectures
: Whitespace-separated list of machine architecture names allowed by the test. If empty or not defined, the test is allowed to run on any machine architecture.

allowed_platforms
Whitespace-separated list of machine platform names allowed by the test. If empty or not defined, the test is allowed to run on any machine platform.

custom.NAME
Custom variable defined by the test where ‘NAME’ denotes the name of the variable. These variables are useful to tag your tests with information specific to your project. The values of such variables are propagated all the way from the tests to the results files and later to any generated reports.

Note that if the name happens to have dashes or any other special characters in it, you will have to use a special Lua syntax to define the property. Refer to the *EXAMPLES* section below for clarification.

description
Textual description of the test.

is_exclusive
If true, indicates that this test program cannot be executed along any other programs at the same time. Test programs that affect global system state, such as those that modify the value of a sysctl(8) setting, must set themselves as exclusive to prevent failures due to race conditions. Defaults to false.

required_configs
Whitespace-separated list of configuration variables that the test requires to be defined before it can run.

required_disk_space
Amount of available disk space that the test needs to run successfully.

required_files
Whitespace-separated list of paths that the test requires to exist before it can run.

required_memory
Amount of physical memory that the test needs to run successfully.

required_programs
Whitespace-separated list of basenames or absolute paths pointing to executable binaries that the test requires to exist before it can run.

required_user
If empty, the test has no restrictions on the calling user for it to run. If set to ‘unprivileged’, the test needs to not run as root. If set to ‘root’, the test must run as root.

timeout
Amount of seconds that the test is allowed to execute before being killed.

Recursion

To reference test programs in another subdirectory, a different **Kyuafile** must be created in that directory and it must be included into the original **Kyuafile** by means of the **include**() function.

include() may only be called with a relative path and with at most one directory component. This is by design: Kyua uses the file system structure as the layout of the test suite definition. Therefore, each subdirectory in a test suite must include its own **Kyuafile** and each **Kyuafile** can only descend into the **Kyuafile**s of immediate subdirectories.

If you need to source a **Kyuafile** located in disjoint parts of your file system namespace, you will have to create a ‘shadow tree’ using symbolic links and possibly helper **Kyuafile**s to plug the various subdirectories together. See the *EXAMPLES* section below for details.

Note that each file is processed in its own Lua environment: there is no mechanism to pass state from one file to the other. The reason for this is that there is no such thing as a "top-level" **Kyuafile** in a test suite: the user has to be able to run the test suite from any directory in a given hierarchy, and this execution must not depend on files that live in parent directories.

Top-level Kyuafile

Every system has a top directory into which test suites get installed. The default is */usr/tests*. Within this directory live test suites, each of which is in an independent subdirectory. Each subdirectory can be provided separately by independent third-party packages.

Kyua allows running all the installed test suites at once in order to provide comprehensive cross-component reports. In order to do this, there is a special file in the top directory that knows how to inspect the subdirectories in search for other Kyuafiles and include them.

The *FILES* section includes more details on where this file lives.

Helper functions

The ‘base’, ‘string’, and ‘table’ Lua modules are fully available in the context of a **Kyuafile**.

The following extra functions are provided by Kyua:

string **current_kyuafile**()
: Returns the absolute path to the current **Kyuafile**.

string **fs.basename**(*string path*)
: Returns the last component of the given path.

string **fs.dirname**(*string path*)
: Returns the given path without its last component or a dot if the path has a single component.

bool **fs.exists**(*string path*)
: Checks if the given path exists. If the path is not absolute, it is relative to the directory containing the **Kyuafile** in which the call to this function occurs.

iterator **fs.files**(*string path*)
: Opens a directory for scanning of its entries. The returned iterator yields an entry on each call, and the entry is simply the filename. If the path is not absolute, it is relative to the directory containing the **Kyuafile** in which the call to this function occurs.

is_absolute **fs.is_absolute**(*string path*)
: Returns true if the given path is absolute; false otherwise.

join **fs.join**(*string path*, *string path*)
: Concatenates the two paths. The second path cannot be absolute.

FILES

/usr/tests/Kyuafile.
: Top-level **Kyuafile** for the current system.

/usr/share/examples/kyua/Kyuafile.top.
: Sample file to serve as a top-level **Kyuafile**.

EXAMPLES

The following **Kyuafile** is the simplest you can define. It provides a test suite definition and registers a couple of different test programs using different interfaces:

```
syntax(2)

test_suite('first')

atf_test_program{name='integration_test'}
```

```
plain_test_program{name='legacy_test'}
```

The following example is a bit more elaborate. It introduces some metadata properties to the test program definitions and recurses into a couple of subdirectories:

```
syntax(2)

test_suite('second')

plain_test_program{name='legacy_test',
                   allowed_architectures='amd64 i386',
                   required_files='/bin/ls',
                   timeout=30}

tap_test_program{name='privileged_test',
                 required_user='root'}

include('module-1/Kyuafile')
include('module-2/Kyuafile')
```

The syntax to define custom properties may be not obvious if their names have any characters that make the property name not be a valid Lua identifier. Dashes are just one example. To set such properties, do something like this:

```
syntax(2)

test_suite('FreeBSD')

plain_test_program{name='the_test',
                   ['custom.FreeBSD-Bug-Id']='category/12345'}
```

Connecting disjoint test suites

Now suppose you had various test suites on your file system and you would like to connect them together so that they could be executed and treated as a single unit. The test suites we would like to connect live under */usr/tests*, */usr/local/tests* and *~/local/tests*.

We cannot create a **Kyuafile** that references these because the **include**() directive does not support absolute paths. Instead, what we can do is create a shadow tree using symbolic links:

```
$ mkdir ~/everything
$ ln -s /usr/tests ~/everything/system-tests
$ ln -s /usr/local/tests ~/everything/local-tests
$ ln -s ~/local/tests ~/everything/home-tests
```

And then we create an *~/everything/Kyuafile* file to drive the execution of the integrated test suite:

```
syntax(2)

test_suite('test-all-the-things')

include('system-tests/Kyuafile')
include('local-tests/Kyuafile')
include('home-tests/Kyuafile')
```

Or, simply, you could reuse the sample top-level **Kyuafile** to avoid having to manually craft the list of directories into which to recurse:

```
$ cp /usr/share/examples/kyua/Kyuafile.top ~/everything/Kyuafile
```

SEE ALSO

kyua(1)