NAME

ots−build − bash library to build OTS packages

SYNTAX

```
# load library
if ots_build=$(pkg−config −−libs ots−build); then
  . $ots_build
else
  echo >&2 "unable to load ots−build library"
  exit 1
fi

# override some default variables
otsb_set var1 val1
otsb_set var2 val2

# initialize the library
otsb_init

# parse the command line
otsb_options "$@"

# create build variables
otsb_set_build_variables

otsb_run_cmds
```

DESCRIPTION

Sometimes the Off-The-Shelf (OTS) software you need has a complicated build workflow. If you want to manage that in a scripted (patch)−config−build−test−install environment, that usually means building a script to perform all of the work. If you are working with dozens of OTS package, that can be a lot of redundant work.

ots-build is a **bash** library which provides support for lots of common tasks in building OTS software. It provides a framework for command line arguments and for running partial or complete builds. It provides a means of patching the package prior to configuration. It provides a rudimentary logging mechanism to keep things tidy.

Because **ots-build** is a library, you'll still have to write a main build script to perform the build, but it should be much simpler.

Invoking the build script

The main build script (which uses **ots-build**) should be invoked with an absoulte path inside the directory containing the extracted package source code. For example, if the build script for package *foo* is located in */imports/foo/build* and the package has been extracted and exists in the directory */tmp/foo−1.23*, then the build script should be invoked as

```
cd /tmp/foo−1.23
/imports/foo/build
```

USING THE LIBRARY

Sharing information with the library

In order to simplify the interface, much of the information needed and provided by **ots-build** is shared via **bash** variables. Variables associated with particular tasks (for instance parsing the command line) are described in the documentation for those tasks.

These variables may be overridden after the library is loaded (using the `otsb_set` function); their documentation will specify when it is safe to do so.

Loading the Library

The library must be loaded *at the very start* of the build script. **ots-build** provides a **pkg-config** metadata file, so may be loaded via

```
if ots_build=$(pkg-config --libs ots-build); then
  . $ots_build
else
  echo >&2 "unable to load ots-build library"
  exit 1
fi
```

Customizing variables

Certain variables must be customized before invoking `otsb_init`; see the documentation for the variables to determine which are affected.

Initializing the Library

ots-build must be initialized by calling the `otsb_init` function.

Parsing Command Line Options

The `otsb_options` function is used to parse command line options and arguments. It takes a list of tokens to parse. Typically this is just those passed on the command line to the script:

```
otsb_options "$@"
```

The quotes are required to maintain proper tokenization. Note that `$@` is set to a function's arguments within a function, so the above invocation will only work in the main body of the **bash** script.

Prior to issuing this command, additional command line options may be added using `otsb_add_option`. The following build options are provided by default:

```
--exec-prefix
--prefix
```

Option format

Only long options (preceded by `--`) are recognized. Options which take values may be separated from their values be a `=` character or white space. For example:

```
--exec-prefix=a --prefix b  --flag
```

Hyphens in option names may be specified as underscores.

Command line options are either boolean (presence signifies true or false), scalars (take a single value) or arrays (may be specified multiple times, appending each value to an array). Flag options may be specified either as affirmative (`--flag`) or negative (`--no-flag`).

Retrieving option values and setting defaults

After parsing, the values of the options are stored in bash variables with the same name as the option (with any hyphens converted to underscores). Boolean values are represented as the string `true` for true and `false` for false. Options which are not specified on the command line are not set to any default value.

Default values for options created via `otsb_add_option` may be set with `otsb_set` before calling `otsb_options`.

In addition to the options that you provide, **ots-build** provides some of its own. These are stored in variables with a special prefix, `otsbv_` to prevent confusion. To specify defaults, use the `otsbv_set` command prior to calling `otsb_init`. For example, the `--logsingle` option value will be stored in `otsbv_logsingle`. To set its default value:

```
otsbv_set logsingle true
```

Option help

ots-build supports a `--help` option.

Options added via calls to `otsb_add_option` will use the help string provided in those calls.

Options provided by **ots-build** have pre-defined help strings which can be overridden by calling the `otsb_help` function prior to calling `ots_init`.

Arguments

The only arguments accepted on the command line are the names of commands. These are specified in the `otsb_commands` array, which defaults to

```
otsb_commands=( patch configure build test install )
```

The actual requested commands are made available in the `otsb_req_cmds` array after the command line is parsed. See ‘‘Commands’’ for more information on how to interface build commands to the auto-invocation code.

Setting build variables

After invoking `otsb_options` and before running commands, build variables must be set by calling `otsb_set_build_variables`.

The `prefix` variable must already be defined, which is typically set via the `--prefix` command line option.

The following variables are set; they may be overriden prior to invoking `otsb_set_build_variables` using `otsb_set`.

import_root
: This is the directory which contains the build script. By default this is extracted from the name of the script (hence the requirement to invoke the build script with an absolute path).

package
: This is the name of the package (including version information). This is by default extracted from the name of the *current directory* using the `otsb_pkg` function.

package_name
: The name of the package (without any version information). This is by default derived from the `package` variable using the `otsb_pkg_name` function. An alternative to overriding the variable is to redefine the `otsb_pkg_name` function.

package_version
: The version of the package. This is by default derived from the `package` variable using the `otsb_pkg_version` function. An alternative to overriding the variable is to redefine the `otsb_pkg_name` function.

patchfile
: The name of the patch file (if any). This is by default set to

```
$import_root/$package.patch
```

exec_prefix
: This will default to the value of `prefix` if it not already set.

bindir
: Defaults to

```
${exec_prefix}/bin
```

libdir
: Defaults to

```
${exec_prefix}/lib
```

incdir
: Defaults to

```
${prefix}/include/${package}
```

mandir
: Defaults to

```
${prefix}/man
```

docdir
: Defaults to

```
${prefix}/share/doc/$package
```

Commands

The main purpose of **ots-build** is to run commands. The commands to run are specified on the command line (see ‘‘Arguments’’). The commands which will always be available are:

all Run all of the commands in the `otsb_commands` array, in order specified.

dump
: Output the command line arguments.

show-cmds
: Output a list of the non-administrative (i.e. actual build workflow) commands to stdout.

ots-build provides the `otsb_run_cmds` function which will execute the commands specified on the command line. A command is implemented as a bash function of the same name, prefixed with *otsb_cmd_*. For example, the *install* command might be written as:

```
otsb_cmd_install () {
  ... commands to perform the actual install ...
}
```

It is important that these functions either exit or return with non-zero values if an error ocurred (see ‘‘Errors’’ for other means of signalling an error).

Stub functions are provided for the `configure`, `build`, `test`, and `install` commands. These will print an error message and exit with a non-zero status (to remind you that you haven’t implemented them properly). You must provide replacement versions of these.

The `patch` command is more fully implemented. See ‘‘Patching’’ for more information.

Patching

ots-build provides a default **patch** command for patching a source distribution before it is configure. This can be changed by redefining the `otsb_cmd_patch` function.

The default function requires the **applypatch** command and a working version of **patch** (Solaris’ version is by definition broken). The `--patch-cmd` option is available to specify an alternative version of **patch**. You can override it via `otsbv_set` before calling `otsb_init`

If the patch file specified via the `patchfile` global variable (see ‘‘ Sharing information with the library’’) exists, it is applied using

```
applypatch -patch "patch -p0 -N -s" $patchfile
```

Logging

If logging is turned on via the `--log` or `--logsingle` options, the output of each command is logged.

`--log` logs each command to a separate file, *pfxcmd*.log, where *pfx* is a prefix specified by `--logpfx`. This defaults to `otsb_`, so that it does not interfere with any log files created by the package (such as *config.log*, etc.).

`--logsingle` sends the output from all commands to a single file, *pfx*.log, where *pfx* is the value of the `--logpfx` option, with any trailing underscore removed.

When logging is turned on, by default no output is sent to the terminal. To log to the terminal as well, specify `--logconsole`.

Defaults for the variables associated with these options may set via the `otsbv_set` function *before* calling `otsb_init`.

Errors

If a command function returns a non-zero value **ots-build** will exit with an error. Typically it is up to the writer of that function to ensure that an error will cause a non-zero return. That can be a pain. **ots-build** provides a simple interface to the **bash** error trapping system which will ensure that if any command executed within a command function exits with an error it will be caught and **ots-build** will be notified.

The `otsb_trap_err` command is used to manage the trap.

To engage or disengage the error trap for all command functions, use

```
otsb_trap_err functions on
otsb_trap_err functions off
```

This sets the default state when a command function is invoked. If this is invoked within a command function it will change the state immediately.

To engage or disengage the error trap immediately

```
otsb_trap_err on
otsb_trap_err off
```

These turn the system on and off immediately. If invoked within a command function, the trap state will return to *off* after the function returns.

To reset the state to that specified for command functions

```
otsb_trap_err functions reset
```

This is only valid within a command function and resets the state to the default state specified by the last `otsb_trap_err functions` setting.

FUNCTIONS

otsb_add_option

```
otsb_add_option $type $name "$help"
```

Add an option of the given type (`scalar`, `bool`, `array`), name, and help string.

otsb_add_dumpvar

```
otsb_add_dumpvar $var_name
```

Add the *named* variable to the list of variables output by the `dump` command. Option variables are automatically added to the list.

otsb_append_path

```
otsb_append_path $path_var_name $path1 $path2 ...
```

Append the specified path(s) to the path variable with the given name. A path variable's value is a colon separated list of paths. For example,

```
foo=a:b:c:d
otsb_append_path foo e f g h
```

results in

```
foo=a:b:c:d:e:f:g:h
```

otsb_prepend_path

```
otsb_prepend_path $path_var_name $path1 $path2 ...
```

Prepend the specified path(s) to the path variable with the given name. A path variable's value is a colon separated list of paths. For example,

```
foo=a:b:c:d
otsb_prepend_path foo e f g h
```

results in

```
foo=e:f:g:h:a:b:c:d
```

otsb_die

```
otsb_die $message
```

Print the message to the standard error stream and exit with a non-zero status code.

otsb_is_true
otsb_is_false

```
otsb_is_true $value
otsb_is_false $value
```

These returns true if the passed value matches what **ots-build** understands as true or false:

```
true:  the strings 'true' or 'on', or a non-zero number
false: the strings 'false' or 'off', or zero
```

Any other values will return false.

otsb_is_boolean

```
otsb_is_boolean $value
```

Returns true if the value is a boolean recognized by either `otsb_is_true` or `otsb_is_false`.

otsb_cleanup

This function is called when the script exits and performs any necessary cleanup. By default it does nothing. You should redefine it if you need it.

otsb_help

```
otsb_help cmd    cmd_name "cmd help string"
otsb_help option option_name "option helpstring"
```

Change the help string of an existing option.

The second argument is either `cmd` or c<option> indicating what element the help string is for. For example, here's how the default help string for the `--exec-prefix` option is specified:

```
otsb_help option exec-prefix  \
   "install architecture-dependent files here"
```

otsb_pkg_name

This function determines the name of the package and prints it to the standard output stream. It is used to generate the value of the `package_name` variable. The default definition is

```
otsb_pkg_version () { echo ${package%%-*} ; }
```

It may be overriden (before calling `ots_set_build_variables`). When called, the `package` variable will have been set.

otsb_pkg_version

This function determines the version of the package and prints it to the standard output stream. It is used to generate the value of the `package_version` variable. The default definition is

```
otsb_pkg_version () { echo ${package##*-} ; }
```

It may be overriden (before calling `ots_set_build_variables`). When called, the `package` variable will have been set.

otsb_run_cmds

This executes the build commands listed in the `otsb_req_cmds` variable, which is filled in by `otsb_options`. The commands are run in the order in which they were specified, *not* the order listed in `otsb_commands`. Do *not* invoke this command more than once.

If a command does not return a successful exit status code, the subsequent commands are not run and `otsb_run_cmds` returns the exit status code of the failing command.

otsb_options

```
otsb_options "$@"
otsb_options "$my_options_from_somewhere_else"
```

`otsb_options` parses the passed tokens for options and commands. It recognizes options specified via the `otsb_options_bools`, `otsb_options_scalars` and `otsb_options_scalars` variables and commands specified via the `otsb_cmds_args` variable. It places values for passed options in variables of the same name and places the requested commands in the `otsb_req_cmds` variable. See ‘‘Parsing the Command Line’’ for more information.

Default values for options may be specified by setting the appropriate option variables before calling `otsb_options`.

otsb_set

```
otsb_set variable value
```

If the named (scalar!) variable has not been set, assign it the specified value:

```
otsb_set prefix "a prefix is here"
```

In most cases this is the preferred way of setting a variable’s value, as it allows using environment variables to specify default values.

For example, if the build script is run in an environment with the environment variable prefix set:

```
prefix=/my/prefix /imports/package/build all
```

then setting the default value of prefix via

```
[...]
prefix=/a/default/value
otsb_options "$@"
```

will ignore the environment, while

```
[...]
otsb_set prefix /a/default/value
otsb_options "$@"
```

will not.

otsb_setarr

```
otsb_set variable value1 value2 ...
```

If the named (array!) variable has not been set, assign it the specified value:

```
otsb_set config_opts --do-this --do-that
```

will do the equivalent of

```
config_opts=( --do-this --do-that )
```

if `config_opts` is empty.

otsbv_set

```
otsbv_set variable value
```

Similar to `otsb_set`, but used only for option variables provided by **ots-build**.

```
otsbv_set log true
```

otsb_set_build_variables

This function is called after all invocations of `otsb_options`. It generates important build variables; see ‘‘Setting build variables’’.

otsb_trap_err
See the ‘‘Errors’’ section.

LOCAL WRAPPERS

If a number of packages require the same boilerplate initialization for **ots-build**, it is possible to redefine some of **ots-build**’s core routines to provide customization; this can be done in a library.

The routines which may be overriden,

```
otsb_init
otsb_options
otsb_set_build_variables
```

are simple wrappers around the actual routines (with an underscore in front of them):

```
otsb_init () {
  _otsb_init
}
```

For example, if all packages should have the `--ots-root` option which is used to figure out where to install the package, redefine `otsb_init` and `otsb_set_build_variables` as follows:

```
. $(pkg-config --libs ots-build)

otsb_init () {

  otsb_options_scalars+=( ots-root platform )
  otsb_help option ots-root "top-level directory into which OTS packages will be i
  otsb_help option platform "platform name"

  _otsb_init
}

otsb_options () {

  _otsb_options "$@"

  otsb_set ots_root /my/default/ots_root

}

otsb_set_build_variables {

  otsb_set prefix $ots_root/$package
  otsb_set platform $(uname -m)
  otsb_set exec_prefix "$prefix/$platform"

  _otsb_set_build_variables
}
```

There’s a small subtlety in the `otsb_options` redefinition. In order for the caller to use the value of `ots_root` to override variables set by `otsb_set_build_variables` its value must be set between calls to `_otsb_options` and `otsb_set_build_variables` and before the caller’s calls to `otsb_set`.

(Written in modern **bash** which supports appending to arrays with +=).

Save the code in *local-ots-build* and load it instead of **ots-build** and all packages will get the extra functionality.

VARIABLES

The following variables are important to **ots-build**. Be careful that you do not inadvertently use them in a different context.

otsb_commands
: An array containing the available commands, in the order that they should be invoked.

otsb_req_cmds
: An array containing the commands requested on the command line.

otsb_options_bools
: An array containing a listing of boolean options.

otsb_options_scalars
: An array containing a listing of options taking a single value.

otsb_options_arrays
: An array containing a listing of options which may be specified multiple times, with each value appended to an array.

otsb_version
: The version of the **ots-build** library.

import_root
: The name of the directory containing the build script, used to find the optional patch file.

package
: The name of the package, including the version number, derived from the directory that the build script is run in.

patchfile
: The (optional) file containing the patch to apply to the source directory, derived from the package name and the directory containing the build script.

version
: The package version, determined from the package name.

log
: A boolean indicating whether or not logging is enabled.

logpfx
: The prefix appended to the command name to create a name for the log file.

logsingle
: A boolean indicating that all logs should go to a single file.

logconsole
: A boolean indicating that logging should be sent to the console as well as to the log files.

COPYRIGHT AND LICENSE

AUTHOR

Diab Jerius <djerius@cfa.harvard.edu>