NAME

mstbuild - manage the build environment and process for MST software

SYNOPSIS

mstbuild [<config options>] [<global options>] command [<command options>] [command [<command options>]]

DESCRIPTION

mstbuild is primarily a front end to a variety of GNU autotool packages, with additional functionality to implement local build conventions. It runs a subset of commands for initializing, bootstrapping, configuring, and making an autotooled package.

It provides sensible, host-specific defaults for compilation flags as well as installation directories. It provides these to the autotools through both command line arguments and environmental variables.

OPTIONS AND ARGUMENTS

mstbuild takes a list of global options followed by one or more commands, which may have options of their own.

Options are specified using a getopt style interface. Long names are available when preceded with double hyphens `--`, in which case only the minimal number of characters are required. Options which take values may be separated from those values with the = character.

Environmental variables may be used to specify additional arguments to the command line:

- `MSTBUILD_ARGS0` specifies **configuration options**. Its contents are processed before any other command line arguments
- `MSTBUILD_ARGS` specifies other types of options. Its contents are processed *after* any configuration options and *before* any other arguments on the command line.

In the following descriptions, `$ROOT` refers to the value of the **--root** argument.

Configuration Options

These options must occur at the very beginning of the command line.

--cwd *dir*

Change to the specified directory before running **mstbuild**.

--configfile|--cfg *file*

The name of the configuration file to load. This defaults to *mstbuild.cfg*.

--buildtarget *target*

The hardware/software platform to compile for. The default is the **sysarch** platform designated as `platform_os_generic`. The target name may be a symbolic name recognized by **sysarch** (e.g. `platform_generic`, etc).

Global Options

--config *path*

The path to a local **autoconf** site configuration file. This defaults to either the environmental variable `CONFIG_SITE`, or `/proj/axaf/share/config.site`, in that order. The value may contain strings of the form `$variable`, which will be replaced by the value of the variable. Available variables are `buildtarget` (which will be replaced by the value of the **--buildtarget** option) as well as any current environmental variable.

--define|**-D** *VAR=value*

Define the specified environmental variable, setting it equal to the specified value. The value of the **--nouenv** option has no affect on this option. The value may contain variable references of the form `${var}`; these will be interpolated by **mstbuild**. The available variables are those generated by the following command:

```
mst_envs --nofullpath -ldlibrarypath
```

The value of the following options are also interpolated

```
root
buildtarget
prefix
exec_prefix
destdir
perl_archname  (Perl Config variable)
```

This option may be specified more than once. See also the **--pdefine** option.

--localdir

This option indicates that all steps after `bootstrap` are run in a separate *local* directory. The directory used is

```
mstlocaldir/package/buildtarget
```

where *mstlocaldir* is the output from the **mstlocaldir** program, *package* is the name of the package, and *buildtarget* is the value of the **--buildtarget** option. This option forces **--target_link** to be true. It has no effect on the state of the **--target** option. See **--target_dir** for more control over this behavior.

If **mstlocaldir** is not available, this option is disabled; **mstbuild** won't even know it exists.

--noarch

This indicates that the package is not architecture specific, and sets

```
exec_prefix = prefix
```

It overrides any command line attempts to specify `exec_prefix`.

--nomstenv

Normally **mstbuild** will use the **MST_Envs** package (if it is installed) to find dependencies in the "standard" places in the MST environment. This option instructs **mstbuild** to not use the package.

--nouenv

Indicate that certain environmental variables will be ignored. See *ENVIRONMENTAL VARIABLES* for the list.

--pdefine|**-P** *VAR*=[**position**]*value*

Append or prepend to the specified environmental variable *VAR*, which is assumed to be a colon delimited list of paths. By default (if **position** is not specified) the value is put at the end of the list. **position** may either by the character + (append) or - (prepend).

The value of the **--nouenv** option has no affect on this option. The value may contain variable references of the form `${var}`; these will be interpolated by **mstbuild**. The available variables are those generated by the following command:

```
mst_envs --nofullpath -ldlibrarypath
```

The value of the following options are also interpolated

```
root
buildtarget
prefix
exec_prefix
destdir
perl_archname  (Perl Config variable)
```

This option may be specified more than once. See also the **--define** option. This option is processed after th **--define** option.

--target|-t

If specified, all steps after `bootstrap` are run in a separate directory given by **--target_dir** (see also **--localdir**). This defaults to on. Specify **--notarget** to keep the build in the current directory.

--target_dir *dir*

If **--target** is specified, this option provides the name of the subdirectory. It defaults to the value of the **--buildtarget** option.

--target_link

If **--target** is specified, create a symbolic link (named after the value of the **--buildtarget** option) from the target directory to the current directory. This defaults to true if **--target** and **--target_dir** are set by the user.

--Target|-T

This is the similar to the **--target** option, except that the subdirectory will be emptied before being used.

--dryrun|-n

If specified once, **mstbuild** will echo the commands it would normally perform, but will not actually execute them. If specified twice, any command which recognizes a "dry-run" option will be executed with that option.

--makeflags|-m *make flags*

This option allows one to specify additional flags to be passed to the **make** command. Use this with extreme caution! **-m** may be repeated.

--man|--usage

print full instructions and exit

--help

print an abbreviated help message and exit

--version

print version and exit

Installation and Grafting options

Values for the following options may include strings of the form `$variable`

```
root prefix exec_prefix
```

These strings will be replaced by the value of the corresponding variable, which may be an environmental

variable or the value of the similarly named option. Values from the following options are available:

```
buildtarget root prefix exec_prefix
```

In order to reduce confusion, the options are processed in the above order, so that one can write the following:

```
--prefix=${root}
```

and get a sensical answer.

--graft *path*

The path to the **graft** executable. This defaults to what was discovered when **mstbuild** was built. If **graft** is not available, the package will not be configured with grafting in mind. Set this to `no` or `off` to force grafting off if it defaults to on.

--root *path*

The root directory off of which all destination paths originate. This defaults to the `${AXAF_SIMUL_ROOT}`. This also defines where **pkg-config** will look for dependencies.

--destdir *path*

Sets the `DESTDIR` variable for **make install** if grafting is on. It defaults to `$ROOT/pkgs`. Do not set this unless you are very brave.

--prefix *prefix*

With grafting on, sets the `prefix` variable for **make install**. With grafting off, sets the `--prefix` option for **configure**. Do not set this unless you are very brave.

--exec_prefix *path*

With grafting on, sets the `exec_prefix` variable for **make install**. With grafting off, sets the `--exec-prefix` option for **configure**. Do not set this unless you are very brave.

Template Options

Some commands take options which modify variables available to the template files. These options are:

--bugreport *string*

This specifies the value for the template `bugreport` variable. It defaults to that in *templates.cfg*.

--copyrightholder *string*

This specifies the value for the template `copyrightholder` variable. It defaults to that in *templates.cfg*.

--emailcontact *string*

This specifies the value for the template `emailcontact` variable. It defaults to that in *templates.cfg*.

--license_type *license*

Specify the license to copy. The available licenses are:

```
GPL2
GPL3
SAO
```

The default is `GPL3`.

--suffix *string*

A suffix to apply to the filenames of all files created by the **chkstds** or **init** options. Directory names are unaffected. A preceding `.` is *not* automatically added.

--template_dir *dir*

Where to copy the templates from. To use this requires more information than is given here.

--version *string*

This specifies the value for the template `version` variable. The default vaule is read from *configure.ac* (if that exists) or is `1.0.0`.

Commands

Commands may have options; these are of the same form as the global options. A command's options directly follow it on the command line. Some commands are conglomerations of multiple commands and accept options for all of the commands they perform. For example, **bootstrap** also runs **snippets**, so accepts the **--copy** option. If some of these commands have similarly named options, this can cause confusion. For example, if a conglomerate command **c** runs commands **a** and **b** and both have an option named **--args**, then in the following command line,

```
c --args foo
```

it's uncertain whether **a** or **b** gets the `--args foo` option. To solve this problem, conglomerate command options with the same names must be prefixed with their command names. For example,

```
c --a-args foo --b-args goo
```

Typically commands are run in the following sequence:

```
bootstrap
configure
make
check
dist
distcheck
install
graft
export
```

To see the actual shell command lines which will be executed, use the **--dryrun** option.

help

```
help [options] [ command_name | options [ option_type] ]
```

show help.

`help` takes the following options:

--page

Use a pager to show the help information.

It take zero or more arguments:

help

shows the list of commands.

help `command`

shows help for `command`.

help options *[args]*

With no additional arguments, this shows the global options. The following arguments are available:

config

shows the configuration options

template

shows the template options

init

```
init [I<options>] [I<package name>]
```

init creates a skeleton package by copying template files to a directory of the given name (which should be the package name). If the directory does not exist, it is created. If no package name is specified, the skeleton is created in the current directory. If a *configure.ac* file is in the current directory, that is scanned for the package name and version, else the package name is set to that of the current directory.

Options:

Note that `init` also takes template options. See *Template Options* or run

```
mstbuild help options template
```

--force

Force files to be created, even if they already exist. Backups will be created for existant files.

--package_type|-t *package type*

This indicates what type of package is to be initialized. The available choices are:

`perl_prog` - a perl program

`prog` - a compiled (C/C++) program

`lib_onedir` - a compiled library in a single directory

`cxx_lib_onedir` - a C++ compiled library in a single directory

`lib_multidir` - a compiled library split into subsections

snippets

check and optionally update snippets

Options:

--copy

Copy snippets to the local directory if they do not exist locally or are out of date.

--diff|-D

If local snippets are out of date, print their differences.

--exclude *pattern*

Exclude files which match the pattern. This uses a slightly extended version of the standard shell file-matching pattern, where the `*` operator can match `/` characters. For example:

```
--exclude 'templates/*'
--exclude 'snippet_dist/*'
```

This option may be repeated.

--force

Force snippets to be copied from the snippet repository, regardless of their state of up-to-dateness.

--xsnip

If local snippets are out of date, exit with a non-zero value. The **--copy** and **--diff** options are still obeyed if specified.

--dir *directory*

Specify the location of the directory holding the snippets. The default (`/proj/axaf/simul/share/mstbuild/snippets`) should suffice.

boot

alias: bootstrap

run **libtoolize**, **aclocal**, **autoheader**, **automake**, and **autoconf**.

Options:

--aclocal_dir *path list*

This is a colon separated list of directories which contain local **autconf** macros. Note that */proj/axaf/simul/share/aclocal* is automatically searched

--autoconf *alternative autoconf*

This specifies an alternative version of **autoconf** to use. By default it uses what is in the execution path.

--automake *alternative automake*

This specifies an alternative version of **automake** to use. By default it uses what is in the execution path.

--aclocal *alternative aclocal*

This specifies an alternative version of **aclocal** to use. By default it uses what is in the execution path.

--autoheader *alternative autoheader*

This specifies an alternative version of **autoheader** to use. By default it uses what is in the execution path.

--libtoolize *alternative libtoolize*

This specifies an alternative version of **libtoolize** to use. By default it uses what is in the execution path.

--autorun *list of commands*

This option limits the list of commands to run to those given. The following commands are recognizes:

```
autoconf automake aclocal autoheader libtoolize
```

config

alias: configure

run **configure**.

Options:

--args *argument list*

Specify additional arguments to **configure**. This option may be repeated. The value of the following options are interpolated into the arguments

```
root
buildtarget
prefix
exec_prefix
destdir
```

--cache | **--nocache**

Turn caching by configure on or off. It is on by default.

make

run `make`

check

run `make check`

chkstds

chkstds checks for packaging consistency. The following checks are made:

required files in the package:

A license file, either `COPYING` or `LICENSE`, and `README`, `NEWS`, and `INSTALL` files

versioned NEWS and ChangeLog files:

The version number in the `configure.ac` file must appear in the first few lines of `NEWS` and `ChangeLog`.

Options:

Note `chkstds` also takes template options. See *Template Options* or run

```
mstbuild help options template
```

--create

Create missing files from templates. A missing `ChangeLog` file cannot be created. Be sure to edit the `README` file.

dist

This first executes the **chkstds** command, then runs `make dist`

distcheck

This first executes the **chkstds** command, then runs `make distcheck`

export

This command copies the distribution tarball to the export directory. If the **--distcheck** option is specified, **distcheck** is first run. If **distcheck** is not run, the tarball must already exist.

Options:

--dir|-d *directory*

The export directory. This defaults to */proj/axaf/simul/export*.

--distcheck|**--nodistcheck**

Run **distcheck** first, which will create the tarball. This is the default behaviour.

--ok|**--no-ok**

If set (the default) it's OK to generate the distribution tarball. This option is mainly useful in the *mstbuild.cfg* file to prevent generation of a tarball for "proprietary" packages which should never be exported.

install

run `make install`.

graft

run

```
graft -nit ${prefix} ${destdir}/${prefix}
```

if there are any conflicts, exit and allow the user to resolve them. otherwise, run

```
graft -it ${prefix} ${destdir}/${prefix}
```

clean

run `make clean`

distclean

run `make distclean`

maintclean

alias: maintainer-clean

run `make maintainer-clean`

configplus

execute commands: configure + make + check + distcheck

most

execute commands: bootstrap + configure + make + check + distcheck

all

execute commands: bootstrap + configure + make + check + distcheck + install + graft + export

dumpenv

output selected environmental variables.

Options:

--all

Output all of the environmental variables, not just the select few.

--shell|**-s** *shell*

set the shell syntax to be used for the **dumpenv** command. Possible values are: `bash`, `sh` , `ksh`, `csh`, `tcsh`.

It defaults to the contents of the `SHELL` environmental variable.

optvals

output values of the global options after applying any command line switches.

Configuration file

mstbuild reads a configuration file at startup, if available. By default it looks for *mstbuild.cfg* in the current directory, but may be directed elsewhere with the **--configfile** option. Configuration files use an Apache style syntax (for indepth information, see *Config::General*). There are two main constructs:

Option value assignment:

```
option_name = value
```

where *option_name* is the long form of one of the command line options. Boolean options should be given a value of `1` or `0` or `true` or `false`, e.g.

```
arch = false
```

Block declaration:

There are unnamed blocks:

```
<Directive>
  ...
</Directive>
```

and named blocks

```
<Directive name>
</Directive>
```

Blocks may be nested. Option value assignment may take place within blocks.

Global configuration options value assignments are generally made outside of any blocks (see the documentation on `BuildTarget` blocks for an exception).

The following blocks are recognized:

BuildTarget

These named blocks are used to specify options specific to a particular platform. The block names should be specific platform designations (although they may contain glob wildcards (e.g. `x86_64-Linux_Debian*`) and are matched against the **sysarch** generated `platform_generic`, `platform_native`, `os_type` and `OS` variables.

The enclosed values are applied after the options outside of the **BuildTarget** block. `buildtarget` options are applied after OS options. Global option assignments should appear at the outermost level of this block. Options which may be specified only once will override options specified outside the **BuildTarget** block. Options which may be specified multiple times (such as **--define** and **--makeflags**) are *added*.

For example,

```
<BuildTarget athlon_64>
  makeflags = -march athlon
</BuildTarget>
```

would set the global **--makeflags** option for **athlon_64** buildtargets.

define

This block contains values for the **--define** option. The block does not take a name. For example,

```
--define a=b --define foo=goo
```

may be represented as

```
<define>
   a = b
   foo = goo
</define>
```

define blocks may be placed inside **BuildTarget** blocks.

Command

These blocks are named with a **mstbuild** command, and are used to specify command specific options.

For example,

```
<Command snippets>
  snippet_dir  = snippets_dist
  exclude      = templates/*
  exclude      = snippets_dist/*
</Command>
```

would set options for the **snippets** command.

Command blocks may be placed inside **BuildTarget** blocks.

EXAMPLES

To bootstrap, configure, make, make check, make install, and graft a package:

```
mstbuild bootstrap configure make check install graft
```

Magic

This script performs a whole lot of magic in order to ensure that programs and libraries will appear in the *$ROOT* directory structure while actually living in separate, package and version specific directories in *$ROOT/pkgs*.

The packages and libraries are installed in *$ROOT/pkgs*, and then grafted onto *$ROOT*. We have to massage things behind the scenes so that operationally none of the packages knows about *$ROOT/pkgs*, just *$ROOT*.

This is a cursory summary of what happens here. Read the **autoconf**, **automake**, **libtool**, and **pkg-config** docs if you want a more in depth understanding, i.e. this makes you say, 'eh?'.

The **--destdir**, **--exec_prefix**, and **--prefix** options combine to specify where the package will actually be installed. This is where the true magic is...

--destdir specifies the `DESTDIR` in the **autoconf**/**automake** sense, a directory used to stage the install... not the final location, but a step along the way. **--exec_prefix** and **--prefix** provide package specific subdirectories within **--destdir**. So despite the fact that they are handed to `install` in the *prefix* and *exec_prefix* variables, they interact with `DESTDIR` in such a way that they are interpreted as subdirectories of **--destdir**. Even if they are specified with a leading /, i.e. they are absolute pathes, they are still treated as subdirectories of **--destdir**.

Confused? Good. Now we are in the same boat.

ENVIRONMENTAL VARIABLES

The following environmental variables are used, if available:

```
MSTBUILD_ARGS MSTBUILD_ARGS0 CONFIG_SITE
MAKE CC CXX F77 CFLAGS CXXFLAGS FFLAGS
```

The second batch will be ignored if the **--nouenv** flag is specfied.

To dump the environment variables that may be set or modified by this code, use the **dumpenv** command.

PKG_CONFIG_PATH

PKG_CONFIG_PATH is treated specially. If the **MST_Env** module is available, an existing **PKG_CONFIG_PATH** variable is ignored. If the **MST_Env** module is not available, an existing variable is used. The **--define** option may be used to override this behavior; it will replace any previous or generated value of **PKG_CONFIG_PATH**.

To add a path to an existing path, use the **--pdefine** option. This option is processed after the **--define** option.

Glossary

CONFIG_SITE

: **autoconf** site wide configuration file

OS

: the OS

MAKE

: make executable to use.

CC

: C compiler

CXX

: C++ compiler

F77

: Fortran 77 compiler

CFLAGS

: C compiler flags

CXXFLAGS

: C++ compiler flags

FFLAGS

Fortran 77 compiler flags

EXAMPLES

Advanced Usage

Debugging a library using an application

Say that you need to compile against a library which has been compiled with debugging turned on, and which uses **pkg-config**. Let's say it's another **mstbuild** enabled compile.

The scheme is to install the library in a temporary spot, then coax the executable compilation to use it. Let $ROOT be the location of the temporary directory.

1 Build and install the library with the following non-standard **mstbuild** flags:

```
--root=$ROOT --graft=no -D CFLAGS=-g -D FFLAGS=-g
```

This creates a simple directory structure. Because **mstbuild** uses **--root** to find dependencies, if the library requires other libraries, you'll have to explicitly specify the **pkg-config** search directories via the **--pdefine** option:

```
--pdefine PKG_CONFIG_PATH=-$OTHERROOT/lib/pkgconfig
```

2 Build the program, adding the correct path to the **pkg-config** metadata files:

```
--graft=no \
--pdefine PKG_CONFIG_PATH=-$ROOT/lib/pkgconfig
```

If the executable has library dependencies in $OTHERROOT, append the *$OTHERROOT/lib/pkgconfig* to the **--pdefine** option value:

```
--graft=no \
--pdefine \
  PKG_CONFIG_PATH=-$ROOT/lib/pkgconfig:$OTHERROOT/lib/pkgconfig
```

Compile and install against a non-MST tree when MST_Env is available.

Normally **MST_Env** is used if it's available. If, however, you're working with dependencies which are too hot for the standard MST install and which are installed in another directory tree, =mstbuild= should ignore the MST environment. This example assumes that the other tree is complete, that all dependencies are found there and that grafting is not used.

There are the options that should be set (**$ROOT** is the alternate directory tree)

```
--graft=no
--root $ROOT
--nomstenv
--define PKG_CONFIG_PATH=$ROOT/lib/pkgconfig
```

COPYRIGHT AND LICENSE

VERSION

This documents version 3.2.18_01 of **mstbuild**.

AUTHORS

Mike Tibbetts

Diab Jerius <djerius@cfa.harvard.edu>