



OpenEmbedded - devtool



Contents

1. OpenEmbedded - devtool	3
2. How to create a new open embedded layer	24



A quality version of this page, approved on 19 October 2021, was based off this revision.

Contents

1 Overview	4
2 Examples	6
2.1 Add a new application or a new library	6
2.1.1 Goal	6
2.1.2 Way of working (step-by-step)	6
2.1.3 Material	7
2.2 modify an existing application or library managed by a recipe on which you have the ownership	8
2.2.1 Goal	8
2.2.2 Way of working (step-by-step)	8
2.2.3 Material	8
2.3 Create/update an append on application or library	9
2.3.1 Goal	9
2.3.2 Way of working (step by step)	9
2.3.3 Material	10
3 List of devtool commands	11
3.1 Beginning to work on a recipe	11
3.1.1 devtool add	11
3.1.2 devtool modify	12
3.1.3 devtool upgrade	13
3.2 Getting information	14
3.2.1 devtool status	14
3.2.2 devtool search	14
3.3 Working on a recipe in the workspace	15
3.3.1 devtool build	15
3.3.2 devtool rename	15
3.3.3 devtool edit-recipe	16
3.3.4 devtool find-recipe	16
3.3.5 devtool configure-help	17
3.3.6 devtool update-recipe	17
3.3.7 devtool reset	18
3.3.8 devtool finish	19
3.4 Testing changes on target	19
3.4.1 devtool deploy-target	19
3.4.2 devtool undeploy-target	20
3.4.3 devtool build-image	21
3.5 Advanced	21
3.5.1 devtool create-workspace	21
3.5.2 devtool import	22
3.5.3 devtool extract	22
3.5.4 devtool sync	23
3.5.5 devtool export	23



1 Overview

OpenEmbedded is a build system to generate **distributions** via images or to generate a SDK.

A tool is available on OpenEmbedded to step into development with your OpenEmbedded distribution, this tool is named **devtool**.

This tool allows to:

- add new recipe
- create append on actual recipe by creating bbappend file
- update existing bbappend file



Official documentation of devtool: [Using devtool in your sdk workflow \(from www.yoctoproject.org\)](http://www.yoctoproject.org)

Display **devtool** help:

```
$ devtool --help
NOTE: Starting bitbake server...
usage: devtool [--basepath BASEPATH] [--bbpath BBPATH] [-d] [-q]
           [--color COLOR] [-h]
           <subcommand> ...

OpenEmbedded development tool

options:
  --basepath BASEPATH  Base directory of SDK / build directory
  --bbpath BBPATH      Explicitly specify the BBPATH, rather than getting it
                       from the metadata
  -d, --debug          Enable debug output
  -q, --quiet          Print only errors
  --color COLOR        Colorize output (where COLOR is auto, always, never)
  -h, --help          show this help message and exit

subcommands:
  Beginning work on a recipe:
    add                Add a new recipe
    modify             Modify the source for an existing recipe
    upgrade            Upgrade an existing recipe
  Getting information:
    status             Show workspace status
    search             Search available recipes
  Working on a recipe in the workspace:
    build              Build a recipe
    rename             Rename a recipe file in the workspace
    edit-recipe        Edit a recipe file in your workspace
    find-recipe        Find a recipe file in your workspace
    configure-help     Get help on configure script options
    update-recipe      Apply changes from external source tree to recipe
    reset              Remove a recipe from your workspace
    finish             Finish working on a recipe in your workspace
  Testing changes on target:
    deploy-target      Deploy recipe output files to live target machine
    undeploy-target    Undeploy recipe output files in live target machine
    build-image        Build image including workspace recipe packages
  Advanced:
```



```
create-workspace  Set up workspace in an alternative location
import            Import exported tar archive into workspace
extract           Extract the source for an existing recipe
sync             Synchronize the source tree for an existing recipe
export           Export workspace into a tar archive
Use devtool <subcommand> --help to get help on a specific command
```



2 Examples

2.1 Add a new application or a new library

2.1.1 Goal

The goal is to integrate a new application or a new library on your OpenEmbedded build setup.

The application or library doesn't have any recipe available on any layer referenced by your OpenEmbedded build setup.

Devtool can help you to generate recipe in your OpenEmbedded build setup and more specifically in a layer you manage.



Devtool is able to detect some mandatory things to help you to integrate application or library like type of configuration (cmake, autotools) but not to define the parameters to pass for configuration. At least you must manually adapt the recipe to your needs :

- by populating correctly SRC_URI,
- by surcharging configure, compile, install command,
- by specifying the list of files to install on target,
- ...

2.1.2 Way of working (step-by-step)

1. Create a source directory if does not exist (example 'mysources' in build dir)

```
$> mkdir mysources/myapp
```

2. Get the source of application (git clone, extract tarball, ...) and put it in the source directory
3. Add the application to the workspace

```
$> devtool add myapp mysources/myapp
(devtool create on workspace a recipe for myapp, see description of command devtool add.)
```

4. Build it:

```
$> devtool build myapp
```

5. Deploy to the target device build output (via network access)

```
$> devtool deploy-target myapp root@<ip of board>
```

6. Edit source code & repeat 4-5 as necessary
7. Populate the layer with your new recipe

First, make sure to create/enable your custom layer **meta-my-custo-layer**. For that you can refer to [How to create a new open embedded layer](#) article.



```
$> mkdir ../meta-st/meta-my-custo-layer/recipes-custom/myapp
$> cp workspace/recipes/myapp/myapp.bb ../meta-st/meta-my-custo-layer/recipes-custom/myapp
$> mkdir ../meta-st/meta-my-custo-layer/recipes-custom/myapp/myapp
$> cp mysources/myapp/* ../meta-st/meta-my-custo-layer/recipes-custom/myapp/myapp
$> cp <appropriated license file> ../meta-st/meta-my-custo-layer/recipes-custom/myapp
/myapp
```

Note that all copied files into folder `../meta-st/meta-my-custo-layer/recipes-custom/myapp/myapp` **must be added in SRC_URI field of myapp.bb**

Then the new recipe (myapp.bb) must be added inside the custom image you compile

```
$> cd meta-st/meta-my-custo-layer/recipes-samples/images/
```

Open my-custom-image.bb and add this line : `IMAGE_INSTALL += "myapp"`

2.1.3 Material

- content of recipe file

```
$> cat workspace/recipes/myapp/myapp.bb
# Recipe created by recipetool
# This is the basis of a recipe and may need further editing in order to be fully
functional.
# (Feel free to remove these comments when editing.)
#
# WARNING: the following LICENSE and LIC_FILES_CHKSUM values are best guesses - it is
# your responsibility to verify that the values are complete and correct.
LICENSE = "Unknown"
LIC_FILES_CHKSUM = "file://LICENSE;md5=6dc31330b6fcb6a82dea131bf3d3d33"
# No information for SRC_URI yet (only an external source tree was specified)
SRC_URI = ""
DEPENDS += "wayland"
inherit cmake
# Specify any options you want to pass to cmake using EXTRA_OECMAKE:
EXTRA_OECMAKE = ""
```

Red text: information detected automatically by devtool.

- workspace tree

```
$> tree workspace/ mysources/
workspace/
|-- appends
|  |-- myapp.bbappend
|-- conf
|  |-- layer.conf
|-- README
|-- recipes
|  |-- myapp
|  |-- myapp.bb
mysources/
|  |-- myapp
```



2.2 modify an existing application or library managed by a recipe on which you have the ownership



devtool is based on external source (externalsrc) and is not able to work on recipes which are already using external source

2.2.1 Goal

The goal is to update a recipe owned by you.

On the following example, we take the library **libsmaf** on which we would like to apply some modifications.

2.2.2 Way of working (step-by-step)

1. Create a source directory (example 'mysources' in build dir)

```
$> mkdir mysources
```

2. Extract source

```
$> devtool modify -x libsmaf mysources/libsmaf
(option -x: request to devtool to extract the source code and patch it following
the rules available on libsmaf recipe.)
```

3. Edit the code and/or commit your changes on local git repository
4. Build it:

```
$> devtool build libsmaf
```

5. Deploy to the target device build output (via network access)

```
$> devtool deploy-target libsmaf root@<ip of board>
```

6. Edit source code & repeat 3-5 as necessary
7. Write as patches on top of recipe

```
$> devtool update-recipe -a ../meta-st/meta-my-custo-layer libsmaf
```

First, make sure to create/enable your custom layer **meta-my-custo-layer**. For that you can refer to [How to create a new open embedded layer](#) article.

8. Come back to normal (remove the component from workspace)

```
$> devtool reset libsmaf
```

2.2.3 Material

- modification made by devtool on libsmaf.bb



```

$> git diff recipes-security/smaf/libsmaf.bb
diff --git a/recipes-security/smaf/libsmaf.bb b/recipes-security/smaf/libsmaf.bb
index 78d5882..6c04a2f 100644
--- a/recipes-security/smaf/libsmaf.bb
+++ b/recipes-security/smaf/libsmaf.bb
@@ -8,7 +8,9 @@ PV="1.0+git"

inherit autotools gettext pkgconfig

--SRC_URI = "git://github.com/cprieuzeau/libsmaf.git;protocol=https"
+SRC_URI = "git://github.com/cprieuzeau/libsmaf.git;protocol=https \
+          file://0001Someschanges.patch \
+          "
S = "${WORKDIR}/git"

```

Red text: information automatically modified by devtool

```

$> ls recipes-security/smaf/libsmaf
0001-Somes-changes.patch

```

Red text: patch file added by devtool

2.3 Create/update an append on application or library



devtool are based on external source (externalsrc) and are not able to work on recipe which are already using external source

2.3.1 Goal

The goal are to update (or create) an append made for a specific recipe.

On this example, we take the library **pixman** on which we are would like to apply some modification.

2.3.2 Way of working (step by step)

1. Create a source directory if not exist (here mysources on build dir)

```

$> mkdir mysources

```

2. Extract source

```

$> devtool modify -x pixman mysources/pixman

```

3. Edit the code and/or commit your change on local git repository
4. Build it:

```

$> devtool build pixman

```

5. Deploy to target device build output (via network access)



```
$> devtool deploy-target pixman root@<ip of board>
```

6. Edit source code & repeat 3-5 as necessary
7. Write as patches on top of recipe append file

```
$> devtool update-recipe -a ../meta-st/meta-my-custo-layer pixman
```

First, make sure to create/enable your custom layer **meta-my-custo-layer**. For that you can refer to [How to create a new open embedded layer](#) article. Then:

8. Come back to normal (remove the component of workspace)

```
$> devtool reset pixman
```

2.3.3 Material

- content of created append file: pixman.bbappend

```
$> cat meta-st/meta-st-framework/recipes-graphics/xorg-lib/pixman_0.32.6.bbappend
# look for files in the layer first
FILESEXTRAPATHS_prepend := "${THISDIR}/${PN}:"
SRC_URI += "file://0001-NV12-format-support.patch \
           file://0001-Somes-Changes.patch \
           "
```

- tree of append made in your layer

```
$> tree meta-st/meta-st-framework/recipes-graphics/xorg-lib
|-- pixman
|   |-- 0001-NV12-format-support.patch
|   |-- 0001-Somes-Changes.patch
|-- pixman_0.32.6.bbappend
```



3 List of devtool commands

3.1 Beginning to work on a recipe

3.1.1 devtool add

Add a new recipe to the workspace

- Help command:

```
$ devtool add --help
NOTE: Starting bitbake server...
usage: devtool add [-h] [--same-dir | --no-same-dir] [--fetch URI]
                  [--fetch-dev] [--version VERSION] [--no-git]
                  [--srcrev SRCREV | --autorev] [--srcbranch SRCBRANCH]
                  [--binary] [--also-native] [--src-subdir SUBDIR]
                  [--mirrors] [--provides PROVIDES]
                  [recipeName] [srctree] [fetchuri]

Adds a new recipe to the workspace to build a specified source tree. Can
optionally fetch a remote URI and unpack it to create the source tree.

arguments:
  recipeName      Name for new recipe to add (just name - no version,
                  path or extension). If not specified, will attempt to
                  auto-detect it.
  srctree         Path to external source tree. If not specified, a
                  subdirectory of
                  <WORKSPACE_LAYER_PATH>/workspace/sources will be used.
  fetchuri       Fetch the specified URI and extract it to create the
                  source tree

options:
  -h, --help      show this help message and exit
  --same-dir, -s  Build in same directory as source
  --no-same-dir   Force build in a separate build directory
  --fetch URI, -f URI  Fetch the specified URI and extract it to create the
                      source tree (deprecated - pass as positional argument
                      instead)
  --fetch-dev     For npm, also fetch devDependencies
  --version VERSION, -V VERSION  Version to use within recipe (PV)
  --no-git, -g    If fetching source, do not set up source tree as a git
                  repository
  --srcrev SRCREV, -S SRCREV  Source revision to fetch if fetching from an SCM such
                              as git (default latest)
  --autorev, -a   When fetching from a git repository, set SRCREV in the
                  recipe to a floating revision instead of fixed
  --srcbranch SRCBRANCH, -B SRCBRANCH  Branch in source repository if fetching from an SCM
                              such as git (default master)
  --binary, -b   Treat the source tree as something that should be
                  installed verbatim (no compilation, same directory
                  structure). Useful with binary packages e.g. RPMs.
  --also-native  Also add native variant (i.e. support building recipe
                  for the build host as well as the target machine)
```



```
--src-subdir SUBDIR  Specify subdirectory within source tree to use
--mirrors            Enable PREMIRRORS and MIRRORS for source tree fetching
                    (disable by default).
--provides PROVIDES, -p PROVIDES
                    Specify an alias for the item provided by the recipe.
                    E.g. virtual/libgl
```

- Example

```
$ devtool add mylibrary <source path/mylibrary>
```

- Workspace: result of add command

```
$ tree workspace/workspace/
|-- appends
|  |-- mylibrary.bbappend
|  |-- conf
|  |-- layer.conf
|-- README
|-- recipes
|  |-- mylibrary
|     |-- mylibrary.bb
```

.bb: recipe file of mylibrary automatically filled with the information found in mysource/mylibrary directory

.bbappend: recipe file for managing source code of mylibrary via externalsrc class.

3.1.2 devtool modify

Modify the source for an existing recipe and add to the workspace.

- Help command:

```
$ devtool modify --help
NOTE: Starting bitbake server...
usage: devtool modify [-h] [--wildcard] [--extract | --no-extract]
                   [--same-dir | --no-same-dir] [--branch BRANCH]
                   [--keep-temp]
                   recipename [srctree]
```

Sets up the build environment to modify the source for an existing recipe. The default behaviour is to extract the source being fetched by the recipe into a git tree so you can work on it; alternatively if you already have your own pre-prepared source tree you can specify -n/--no-extract.

arguments:

```
  recipename      Name of existing recipe to edit (just name - no
                  version, path or extension)
  srctree         Path to external source tree. If not specified, a
                  subdirectory of
                  <WORKSPACE_LAYER_PATH>//workspace/sources will be used.
```

options:

```
-h, --help      show this help message and exit
--wildcard, -w  Use wildcard for unversioned bbappend
--extract, -x   Extract source for recipe (default)
--no-extract, -n Do not extract source, expect it to exist
```



```
--same-dir, -s      Build in same directory as source
--no-same-dir      Force build in a separate build directory
--branch BRANCH, -b BRANCH
                    Name for development branch to checkout (when not
                    using -n/--no-extract) (default "devtool")
--keep-temp        Keep temporary directory (for debugging)
```

- Example

```
$ devtool modify -x mylibrary <source path/mylibrary>
(option -x: request to devtool to extract the source code and patch it on <source path
/mylibrary>
following the rule available on mylibrary recipe.
```

- Workspace:

```
$ tree workspace
workspace/
|-- appends
|  |-- mylibrary_<version>.bbappend
|-- conf
|  |-- layer.conf
|-- README
-- recipes
```

3.1.3 devtool upgrade

Upgrade the source of an existing recipe to a new version

- Help command:

```
$ devtool upgrade --help
NOTE: Starting bitbake server...
usage: devtool upgrade [-h] [--version VERSION] [--srcrev SRCREV]
                    [--srcbranch SRCBRANCH] [--branch BRANCH] [--no-patch]
                    [--same-dir | --no-same-dir] [--keep-temp]
                    recipeName [srctree]
```

Upgrades an existing recipe to a new upstream version. Puts the upgraded recipe file into the workspace along with any associated files, and extracts the source tree to a specified location (in case patches need rebasing or adding to as a result of the upgrade).

arguments:

```
recipeName          Name of recipe to upgrade (just name - no version,
                    path or extension)
srctree             Path to where to extract the source tree. If not
                    specified, a subdirectory of
                    <WORKSPACE_LAYER_PATH>/workspace/sources will be used.
```

options:

```
-h, --help          show this help message and exit
--version VERSION, -V VERSION
                    Version to upgrade to (PV)
--srcrev SRCREV, -S SRCREV
                    Source revision to upgrade to (required if fetching
                    from an SCM such as git)
--srcbranch SRCBRANCH, -B SRCBRANCH
                    Branch in source repository containing the revision to
                    use (if fetching from an SCM such as git)
```



```
--branch BRANCH, -b BRANCH      Name for new development branch to checkout (default
                                  "devtool")
--no-patch                       Do not apply patches from the recipe to the new source
                                  code
--same-dir, -s                   Build in same directory as source
--no-same-dir                     Force build in a separate build directory
--keep-temp                       Keep temporary directory (for debugging)
```

- Example

```
$ devtool upgrade mylib --srcrev <NEW SHA1>
```

3.2 Getting information

3.2.1 devtool status

Show workspace status (list of recipe(s) on workspace).

- Help command:

```
$ devtool status --help
NOTE: Starting bitbake server...
usage: devtool status [-h]
```

Lists recipes currently in your workspace and the paths to their respective external source trees

```
options:
-h, --help  show this help message and exit
```

- Example

```
$ devtool status
```

3.2.2 devtool search

Search available recipes (same as **bitbake -s** but with matching pattern).

- Help command:

```
$ devtool search --help
NOTE: Starting bitbake server...
usage: devtool search [-h] keyword
```

Searches for available target recipes. Matches on recipe name, package name, description and installed files, and prints the recipe name on match.

```
arguments:
 keyword      Keyword to search for (regular expression syntax allowed)
```

```
options:
-h, --help  show this help message and exit
```

- Example



```
$ devtool search weston
glibc          GLIBC (GNU C Library)
weston         Weston, a Wayland compositor
libgcc         GNU cc and gcc C compilers
gcc-runtime    Runtime libraries from GCC
weston-init    Startup script and systemd unit file for the Weston Wayland
compositor
```

3.3 Working on a recipe in the workspace

3.3.1 devtool build

Build a recipe, request to devtool to perform do_configure, do_compile and do_install for a specific package managed in the workspace.

- Help command:

```
$ devtool build --help
NOTE: Starting bitbake server...
usage: devtool build [-h] [-s] recipename

Builds the specified recipe using bitbake (up to and including
do_populate_sysroot, do_packagedata)

arguments:
  recipename          Recipe to build

options:
  -h, --help          show this help message and exit
  -s, --disable-parallel-make
                      Disable make parallelism
```

- Example

```
$ devtool build myapp
```

3.3.2 devtool rename

Rename the recipe file name with a new name.

- Help command:

```
$ devtool rename --help
NOTE: Starting bitbake server...
usage: devtool rename [-h] [--version VERSION] [--no-srctree]
                    recipename [newname]

Renames the recipe file for a recipe in the workspace, changing the name or
version part or both, ensuring that all references within the workspace are
updated at the same time. Only works when the recipe file itself is in the
workspace, e.g. after devtool add. Particularly useful when devtool add did
not automatically determine the correct name.

arguments:
  recipename          Current name of recipe to rename
  newname             New name for recipe (optional, not needed if you only
                      want to change the version)
```



```
options:
  -h, --help          show this help message and exit
  --version VERSION, -V VERSION
                      Change the version (NOTE: this does not change the
                      version fetched by the recipe, just the version in the
                      recipe file name)
  --no-srctree, -s    Do not rename the source tree directory (if the
                      default source tree path has been used) - keeping the
                      old name may be desirable if there are internal/other
                      external references to this path
```

- Example

```
$ devtool rename mylib mylib_newname
```



This commands only works after usage of 'devtool add' or 'devtool upgrade' command as it needs the recipe file itself to be in the workspace.

3.3.3 devtool edit-recipe

Edit recipe file in the configured editor

- Help command:

```
$ devtool edit-recipe --help
NOTE: Starting bitbake server...
usage: devtool edit-recipe [-h] [--any-recipe] recipename

Runs the default editor (as specified by the EDITOR variable) on the specified
recipe. Note that the recipe file itself must be in the workspace (i.e. as a
result of "devtool add" or "devtool upgrade"); you can override this with the
-a/--any-recipe option.

arguments:
  recipename          Recipe to edit

options:
  -h, --help          show this help message and exit
  --any-recipe, -a    Edit any recipe, not just where the recipe file itself is
                      in the workspace
```

- Example

```
$ devtool edit-recipe mylib
```

3.3.4 devtool find-recipe

Find any recipe file

- Help command:

```
$ devtool find-recipe --help
NOTE: Starting bitbake server...
usage: devtool find-recipe [-h] [--any-recipe] recipename

By default, this will find a recipe file in your workspace; you can override
this with the -a/--any-recipe option.
```




```
arguments:
  recipename      Recipe to find

options:
  -h, --help      show this help message and exit
  --any-recipe, -a Find any recipe, not just where the recipe file itself is
                  in the workspace
```

- Example

```
$ devtool find-recipe mylib
```

3.3.5 devtool configure-help

Display the configure help for a recipe using such script.

- Help command:

```
$ devtool configure-help --help
NOTE: Starting bitbake server...
usage: devtool configure-help [options] recipename [--arg ...]

Displays the help for the configure script for the specified recipe (i.e. runs
./configure --help) prefaced by a header describing the current options being
specified. Output is piped through less (or whatever PAGER is set to, if set)
for easy browsing.

arguments:
  recipename      Recipe to show configure help for

options:
  -h, --help      show this help message and exit
  -p, --no-pager  Disable paged output
  -n, --no-header Disable explanatory header text
  --arg ...       Pass remaining arguments to the configure script instead of
                  --help (useful if the script has additional help options)
```

- Example

```
$ devtool configure-help mylib
```

3.3.6 devtool update-recipe

Apply changes from external source tree to recipe:

- by updating bb file
- by updating or creating bbappend file

- Help command:

```
$ devtool update-recipe --help
NOTE: Starting bitbake server...
usage: devtool update-recipe [-h] [--mode MODE] [--initial-rev INITIAL_REV]
                             [--append LAYERDIR] [--wildcard-version]
                             [--no-remove]
                             recipename
```



Applies changes from external source tree to a recipe (updating/adding/removing patches as necessary, or by updating SRCREV). Note that these changes need to have been committed to the git repository in order to be recognised.

```
arguments:
  recipename          Name of recipe to update

options:
  -h, --help          show this help message and exit
  --mode MODE, -m MODE Update mode (where MODE is patch, srcrev, auto;
                       default is auto)
  --initial-rev INITIAL_REV
                       Override starting revision for patches
  --append LAYERDIR, -a LAYERDIR
                       Write changes to a bbappend in the specified layer
                       instead of the recipe
  --wildcard-version, -w
                       In conjunction with -a/--append, use a wildcard to
                       make the bbappend apply to any recipe version
  --no-remove, -n    Don't remove patches, only add or update
```

- Example

Create a bbappend for a specific recipe, here pixman

```
$ devtool update-recipe -a ../meta-st/meta-st-framework pixman
```

Update the recipe libsmaf

```
$ devtool update-recipe libsmaf
```

3.3.7 devtool reset

Remove a recipe from your workspace (only on workspace).

- Help command:

```
$ devtool reset --help
NOTE: Starting bitbake server...
usage: devtool reset [-h] [--all] [--no-clean] [recipename [recipename ...]]

Removes the specified recipe(s) from your workspace (resetting its state back
to that defined by the metadata).

arguments:
  recipename          Recipe to reset

options:
  -h, --help          show this help message and exit
  --all, -a           Reset all recipes (clear workspace)
  --no-clean, -n     Don't clean the sysroot to remove recipe output
```

- Example

```
$ devtool reset myapp
```



3.3.8 devtool finish

Allow to complete the development done through devtool by updating layer(s) with the work done.

- Help command:

```
$ devtool finish --help
NOTE: Starting bitbake server...
usage: devtool finish [-h] [--mode MODE] [--initial-rev INITIAL_REV]
       recipename destination
```

Pushes any committed changes to the specified recipe to the specified layer and removes it from your workspace. Roughly equivalent to an update-recipe followed by reset, except the update-recipe step will do the "right thing" depending on the recipe and the destination layer specified.

arguments:

recipename	Recipe to finish
destination	Layer/path to put recipe into. Can be the name of a layer configured in your bblayers.conf, the path to the base of a layer, or a partial path inside a layer. devtool finish will attempt to complete the path based on the layer's structure.

options:

-h, --help	show this help message and exit
--mode MODE, -m MODE	Update mode (where MODE is patch, srcrev, auto; default is auto)
--initial-rev INITIAL_REV	Override starting revision for patches

- Example

```
$ devtool finish mylib Layer/path
```

3.4 Testing changes on target

3.4.1 devtool deploy-target

Deploy recipe output files to live target machine.

- Help command:

```
$ devtool deploy-target --help
NOTE: Starting bitbake server...
usage: devtool deploy-target [-h] [-c] [-s] [-n] [-p] [--no-check-space]
       [-P PORT] [-S | --no-strip]
       recipename target
```

Deploys a recipe's build output (i.e. the output of the do_install task) to a live target machine over ssh. By default, any existing files will be preserved instead of being overwritten and will be restored if you run devtool undeploy-target. Note: this only deploys the recipe itself and not any runtime dependencies, so it is assumed that those have been installed on the target beforehand.

arguments:

recipename	Recipe to deploy
target	Live target machine running an ssh server: user@hostname[:destdir]



```
options:
  -h, --help                show this help message and exit
  -c, --no-host-check       Disable ssh host key checking
  -s, --show-status         Show progress/status output
  -n, --dry-run             List files to be deployed only
  -p, --no-preserve         Do not preserve existing files
  --no-check-space          Do not check for available space before deploying
  -P PORT, --port PORT     Port to use for connection to the target
  -S, --strip               Strip executables prior to deploying (default: False).
                           The default value of this option can be controlled by
                           setting the strip option in the [Deploy] section to
                           True or False.
  --no-strip               Do not strip executables prior to deploy
```

- Example

```
$ devtool deploy-target myapp root@<ip of board>
```



If your package depends on an other package on runtime (RDEPENDS), it will not force the installation of the dependent package in your rootfs. Therefore you will need to use "devtool build-image" to make sure your rootfs is updated.



'deploy-target' copies 'D' directory. It contains all files generated by the recipe. So its content depends on which recipe packages are installed on the target. However 'D' could contain files that do not need to be installed on target.

This is the case in particular for virtual/kernel recipe that generates the file 'vmlinux' which is not installed on target. So by doing a 'deploy-target' with kernel recipe, it could generate some 'no space left' errors (that can be solved by manually removing 'vmlinux' before launching the deploy command)

3.4.2 devtool undeploy-target

Remove recipe output files from live target machine.

- Help command:

```
$ devtool undeploy-target --help
NOTE: Starting bitbake server...
usage: devtool undeploy-target [-h] [-c] [-s] [-a] [-n] [-P PORT]
                               [recipeName] target
```

Un-deploys recipe output files previously deployed to a live target machine by devtool deploy-target.

```
arguments:
  recipeName  Recipe to undeploy (if not using -a/--all)
  target      Live target machine running an ssh server:
              user@hostname
```

```
options:
  -h, --help                show this help message and exit
  -c, --no-host-check       Disable ssh host key checking
  -s, --show-status         Show progress/status output
  -a, --all                 Undeploy all recipes deployed on the target
  -n, --dry-run             List files to be undeployed only
  -P PORT, --port PORT     Port to use for connection to the target
```



- Example

```
$ devtool undeploy-target myapp root@<ip of board>
```



this command may permanently remove the files from the live target machine

3.4.3 devtool build-image

Build image including workspace recipe packages.

- Help command:

```
$ devtool build-image --help
NOTE: Starting bitbake server...
usage: devtool build-image [-h] [-p PACKAGES] [imagenam]

Builds an image, extending it to include packages from recipes in the
workspace

arguments:
  imagenam          Image recipe to build

options:
  -h, --help          show this help message and exit
  -p PACKAGES, --add-packages PACKAGES
                    Instead of adding packages for the entire workspace,
                    specify packages to be added to the image (separate
                    multiple packages by commas)
```

- Example

```
$ devtool build-image st-image-weston
```

3.5 Advanced

3.5.1 devtool create-workspace

Create a specific workspace folder instead of the default one

- Help command:

```
$ devtool create-workspace --help
NOTE: Starting bitbake server...
usage: devtool create-workspace [-h] [--create-only] [layerpath]

Sets up a new workspace. NOTE: other devtool subcommands will create a
workspace automatically as needed, so you only need to use devtool create-
workspace if you want to specify where the workspace should be located.
```



```
arguments:
  layerpath      Path in which the workspace layer should be created

options:
  -h, --help     show this help message and exit
  --create-only  Only create the workspace layer, do not alter configuration
```

- Example

```
$ devtool create-workspace NewWorkspace/path
```

3.5.2 devtool import

Import any exported workspace into the workspace.

- Help command:

```
$ devtool import --help
NOTE: Starting bitbake server...
usage: devtool import [-h] [--overwrite] FILE

Import tar archive previously created by "devtool export" into workspace

arguments:
  FILE          Name of the tar archive to import

options:
  -h, --help     show this help message and exit
  --overwrite, -o Overwrite files when extracting
```

- Example

```
$ devtool import ExportedWorkspaceFileName
```

3.5.3 devtool extract

Extract the source for an existing recipe:

- extract of source code (do_unpack).
- patch it (do_patch).

- Help command:

```
$ devtool extract --help
NOTE: Starting bitbake server...
usage: devtool extract [-h] [--branch BRANCH] [--keep-temp] recipename srctree

Extracts the source for an existing recipe

arguments:
  recipename      Name of recipe to extract the source for
  srctree         Path to where to extract the source tree
```



```
options:
  -h, --help           show this help message and exit
  --branch BRANCH, -b BRANCH
                       Name for development branch to checkout (default
                       "devtool")
  --keep-temp          Keep temporary directory (for debugging)
```

- Example

```
$ devtool extract pixman <source path/pixman>
```

3.5.4 devtool sync

Synchronize extracted sources

- Help command:

```
$ devtool sync --help
NOTE: Starting bitbake server...
usage: devtool sync [-h] [--branch BRANCH] [--keep-temp] recipename srctree

Synchronize the previously extracted source tree for an existing recipe

arguments:
  recipename          Name of recipe to sync the source for
  srctree             Path to the source tree

options:
  -h, --help           show this help message and exit
  --branch BRANCH, -b BRANCH
                       Name for development branch to checkout (default:
                       devtool)
  --keep-temp          Keep temporary directory (for debugging) (default:
                       False)
```

- Example

```
$ devtool sync libsmf mysources/libsmf
```

3.5.5 devtool export

Export the workspace as tarball.

- Help command:

```
$ devtool export --help
NOTE: Starting bitbake server...
usage: devtool export [-h] [--file FILE] [--overwrite]
                    [--include INCLUDE [INCLUDE ...] | --exclude EXCLUDE
                    [EXCLUDE ...]]

Export one or more recipes from current workspace into a tar archive

options:
  -h, --help           show this help message and exit
```



```
--file FILE, -f FILE   Output archive file name
--overwrite, -o       Overwrite previous export tar archive
--include INCLUDE [INCLUDE ...], -i INCLUDE [INCLUDE ...]
                       Include recipes into the tar archive
--exclude EXCLUDE [EXCLUDE ...], -e EXCLUDE [EXCLUDE ...]
                       Exclude recipes into the tar archive
```

- Example

```
$ devtool export ExportWorkspaceFileName
```

Software development kit (A programming package that enables a programmer to develop applications for a specific platform.)

fourcc of YUV420 semi-planar pixel format
 Stable: 19.10.2020 - 14:17 / Revision: 19.10.2020 - 14:16

A quality version of this page, approved on 19 October 2020, was based off this revision.

This article describes how to create and add a new layer to your STM32MPU Embedded Software Distribution Package for any development platform of the STM32MP1 family (for example an STM32MP157 Evaluation board). This can be done in order to modify some of its installed software, or to add new applications. It first explains the OpenEmbedded principle for customization by distribution, and then gives a step-by-step approach to creating a new layer and adding it to your distribution. Finally, it gives guidelines on how to upgrade (add, remove, configure, improve...) any piece of installed software.

Contents

1 Why create a layer?	24
2 How to proceed	25
2.1 Tools	26
2.2 Creating a new layer	27
2.2.1 Available layers from configuration	27
2.2.2 Create the new layer	27
2.2.3 Recommended actions on a new layer	28
2.3 Add the new layer to your configuration	29
2.3.1 Generic case	29
2.3.2 With STMicroelectronics Distribution Package	30

1 Why create a layer?

The OpenEmbedded distribution comes with a set of layers that provide the different pieces of software used to build images.

Checking the layers included in your distribution is easy with the **bitbake-layers** command tool:

- From your top directory, source your BitBake env setup (if not already done):

```
$ source ./layers/meta-st/scripts/envsetup.sh
```

- List your layers and priority:



```
$ bitbake-layers show-layers
layer                path                                     priority
=====
meta                 /local/openstlinux-18-01-23/layers/openembedded-core/meta 5
meta-oe              /local/openstlinux-18-01-23/layers/meta-openembedded/meta-oe 6
...
```

When you want to integrate or modify a distribution , the **OpenEmbedded principle is to append modifications/addons in a dedicated place** outside of the already available distribution layers.

This place is a new layer that you add to the distribution layers and feed with your modifications or add-ons. Hence the first thing to do when you want to integrate your modifications into a distribution is to create a new layer.

2 How to proceed



2.1 Tools

The openembedded-core layer provides a utility tool called **bitbake-layers**, which performs actions associated with layers. It allows you to perform actions including the following:

- see the list of actual layers taken into account in your configuration
- add existing layers to your configuration
- create a new layer

For an exhaustive list of options and subcommands:

- From your top directory, source your BitBake env setup (if not already done):

```
$ source ./layers/meta-st/scripts/envsetup.sh
```

- Display bitbake-layers help:

```
$ bitbake-layers --help
NOTE: Starting bitbake server...
usage: bitbake-layers [-d] [-q] [-F] [--color COLOR] [-h] <subcommand> ...

BitBake layers utility

optional arguments:
  -d, --debug           Enable debug output
  -q, --quiet           Print only errors
  -F, --force           Force add without recipe parse verification
  --color COLOR         Colorize output (where COLOR is auto, always, never)
  -h, --help           Show this help message and exit

subcommands:
  <subcommand>
  show-layers           Show current configured layers
  show-overlaid         List overlaid recipes (where the same recipe exists
                        in another layer)
  show-recipes          List available recipes, showing the layer by which they
                        are provided
  show-appends          List bbappend files and recipe files to which they apply
  show-cross-depends   Show dependencies between recipes that cross layer
                        boundaries
  add-layer             Add a layer to bblayers.conf
  remove-layer          Remove a layer from bblayers.conf
  flatten               flatten layer configuration into a separate output
                        directory
  layerindex-fetch      Fetches a layer from a layer index along with its
                        dependent layers, and adds them to conf/bblayers.conf
  layerindex-show-depends
                        Find layer dependencies from layer index.
  create-layer          Create a basic layer

Use bitbake-layers <subcommand> --help to get help on a specific command
```



2.2 Creating a new layer

2.2.1 Available layers from configuration


You may first check the list of available layers from your configuration and their priority:

- From your top directory, source your BitBake env setup (if not already done):

```
$ source ./layers/meta-st/scripts/envsetup.sh
```

- Then get the list of layers for your configuration:

```
$ bitbake-layers show-layers
layer                                path                                priority
=====
meta-oe                              /local/openstlinux-18-01-23/layers/meta-openembedded/meta-oe 6
meta-gnome                            /local/openstlinux-18-01-23/layers/meta-openembedded/meta-gnome 7
meta-xfce                              /local/openstlinux-18-01-23/layers/meta-openembedded/meta-xfce 7
meta-initramfs                         /local/openstlinux-18-01-23/layers/meta-openembedded/meta-initramfs 8
meta-multimedia                        /local/openstlinux-18-01-23/layers/meta-openembedded/meta-
multimedia 6
meta-networking                       /local/openstlinux-18-01-23/layers/meta-openembedded/meta-
networking 5
meta-webserver                        /local/openstlinux-18-01-23/layers/meta-openembedded/meta-
webserver 6
meta-fileystems                       /local/openstlinux-18-01-23/layers/meta-openembedded/meta-
filesystems 6
meta-perl                              /local/openstlinux-18-01-23/layers/meta-openembedded/meta-perl 6
meta-python                           /local/openstlinux-18-01-23/layers/meta-openembedded/meta-python 7
meta-st-stm32mp                       /local/openstlinux-18-01-23/layers/meta-st/meta-st-stm32mp 6
meta-qt5                               /local/openstlinux-18-01-23/layers/meta-qt5 7
meta-st-openstlinux                   /local/openstlinux-18-01-23/layers/meta-st/meta-st-openstlinux 5
meta                                   /local/openstlinux-18-01-23/layers/openembedded-core/meta 5
```

 priority: indicates the order followed by bitbake to apply rules from layers. When two layers have a '.bbappend' file based on the same recipe, the rules defined by the layer having the lowest priority are applied first, and then the rules defined by the layer having the highest priority are applied. This allows a layer with higher priority than the other layers to override any rules defined by the other layers.

For more information about priority see the documentation: [Priority doc](#)

2.2.2 Create the new layer

To create a new layer you need to specify:

- the location of the new layer
- the name of the new layer
- (optionally) the priority of the new layer.

As an example, let's create the new layer *meta-my-custo-layer* on *meta-st* directory with priority set to 7:



```
$ bitbake-layers create-layer --priority 7 ../layers/meta-st/meta-my-custo-layer
NOTE: Starting bitbake server...
Add your new layer with 'bitbake-layers add-layer ../layers/meta-st/meta-my-custo-layer'
```

```
$ cd ../layers/meta-st
$ tree meta-my-custo-layer
meta-my-custo-layer
├── conf
│   └── layer.conf
├── COPYING.MIT
├── README
├── recipes-example
│   └── example
│       └── example.bb
3 directories, 4 files
```

2.2.3 Recommended actions on a new layer

- Update the README file
- Update the priority in the **conf/layer.conf** file (if it's not the expected one)
- Apply your own modifications and/or addons for the distribution



Note: you can refer to [OpenEmbedded - devtool](#) to learn how to add new applications, modify existing applications and more.



2.3 Add the new layer to your configuration

2.3.1 Generic case

You may first check the list of available layers from your configuration and their priority:

- From your top directory, source your BitBake env setup (if not already done):

```
$ source ./layers/meta-st/scripts/envsetup.sh
```

- Then get the list of layers for your configuration:

```
$ bitbake-layers show-layers
layer                path                                                    priority
=====
meta-oe              /local/openstlinux-18-01-23/layers/meta-openembedded/meta-oe 6
meta-gnome           /local/openstlinux-18-01-23/layers/meta-openembedded/meta-gnome 7
meta-xfce            /local/openstlinux-18-01-23/layers/meta-openembedded/meta-xfce 7
meta-initramfs      /local/openstlinux-18-01-23/layers/meta-openembedded/meta-initramfs 8
meta-multimedia     /local/openstlinux-18-01-23/layers/meta-openembedded/meta-
multimedia 6
meta-networking     /local/openstlinux-18-01-23/layers/meta-openembedded/meta-
networking 5
meta-webserver      /local/openstlinux-18-01-23/layers/meta-openembedded/meta-
webserver 6
meta-fileSystems    /local/openstlinux-18-01-23/layers/meta-openembedded/meta-
filesystems 6
meta-perl            /local/openstlinux-18-01-23/layers/meta-openembedded/meta-perl 6
meta-python         /local/openstlinux-18-01-23/layers/meta-openembedded/meta-python 7
meta-st-stm32mp     /local/openstlinux-18-01-23/layers/meta-st/meta-st-stm32mp 6
meta-qt5             /local/openstlinux-18-01-23/layers/meta-qt5 7
meta-st-openstlinux /local/openstlinux-18-01-23/layers/meta-st/meta-st-openstlinux 5
meta                 /local/openstlinux-18-01-23/layers/openembedded-core/meta 5
```

- You can then add the new layer *meta-my-custo-layer* to your build configuration:



Make sure that you run the command from your build directory

```
$ bitbake-layers add-layer ../layers/meta-st/meta-my-custo-layer/
```

- Then check that it has been properly enabled for your configuration:

```
$ bitbake-layers show-layers
layer                path                                                    priority
=====
meta-oe              /local/openstlinux-18-01-23/layers/meta-openembedded/meta-oe 6
meta-gnome           /local/openstlinux-18-01-23/layers/meta-openembedded/meta-gnome 7
meta-xfce            /local/openstlinux-18-01-23/layers/meta-openembedded/meta-xfce 7
meta-initramfs      /local/openstlinux-18-01-23/layers/meta-openembedded/meta-initramfs 8
meta-multimedia     /local/openstlinux-18-01-23/layers/meta-openembedded/meta-
```



```

multimedia 6
meta-networking /local/openstlinux-18-01-23/layers/meta-openembedded/meta-
networking 5
meta-webserver /local/openstlinux-18-01-23/layers/meta-openembedded/meta-
webserver 6
meta-fileSystems /local/openstlinux-18-01-23/layers/meta-openembedded/meta-
fileSystems 6
meta-perl /local/openstlinux-18-01-23/layers/meta-openembedded/meta-perl 6
meta-python /local/openstlinux-18-01-23/layers/meta-openembedded/meta-python 7
meta-st-stm32mp /local/openstlinux-18-01-23/layers/meta-st/meta-st-stm32mp 6
meta-qt5 /local/openstlinux-18-01-23/layers/meta-qt5 7
meta-st-openstlinux /local/openstlinux-18-01-23/layers/meta-st/meta-st-openstlinux 5
meta /local/openstlinux-18-01-23/layers/openembedded-core/meta 5
meta-my-custo-layer /local/openstlinux-18-01-23/layers/meta-st/meta-my-custo-layer 7

```

2.3.2 With STMicroelectronics Distribution Package

Pre-requisite:

- Your custom layers are integrated in git.

To apply the new layer on top of fresh Distribution Package delivery:

- Download your layer using the *git clone* command under the Distribution Package delivery folder tree
- Source your build environment:

```
$ source ./layers/meta-st/scripts/envsetup.sh
```

- Add your layer *meta-my-custo-layer* to your build configuration

```
bitbake-layers add-layer ../layers/meta-st/meta-my-custo-layer/
```

Initial ramdisk (https://en.wikipedia.org/wiki/Initial_ramdisk)