kas Documentation

Release 0.10.0

Daniel Wagner, Jan Kiszka, Claudius Heine

Jul 24, 2017

Contents

Contents:

CHAPTER 1

Introduction and installation

This tool provides an easy mechanism to setup bitbake based projects.

The OpenEmbedded tooling support starts at step 2 with bitbake. The downloading of sources and then configuration has to be done by hand. Usually, this is explained in a README. Instead kas is using a project configuration file and does the download and configuration phase.

Currently supported Yocto versions:

- 2.1 (Krogoth)
- 2.2 (Morty)

Older or newer versions may work as well but haven't been tested intensively.

Key features provided by the build tool:

- clone and checkout bitbake layers
- create default bitbake settings (machine, arch, ...)
- launch minimal build environment, reducing risk of host contamination
- initiate bitbake build process

CHAPTER 2

User Guide

Dependencies & installation

This projects depends on

- Python 3
- distro Python 3 package
- PyYAML Python 3 package (optional, for yaml file support)

If you need Python 2 support consider sending patches. The most obvious place to start is to use the trollius package intead of asyncio.

To install kas into your python site-package repository, run:

```
$ sudo pip3 install .
```

Usage

There are three options for using kas:

- Install it locally via pip to get the `kas` command.
- Use the docker image. In this case run the commands in the examples below within `docker run -it <kas-image> sh` or bind-mount the project into the container.
- Use the **run-kas** wrapper from this directory. In this case replace `kas` in the examples below with `path/to/run-kas`.

Start build:

```
$ kas build /path/to/kas-project.yml
```

Alternatively, experienced bitbake users can invoke usual **bitbake** steps manually, e.g.:

```
$ kas shell /path/to/kas-project.yml -c 'bitbake dosfsutils-native'
```

kas will place downloads and build artifacts under the current directory when being invoked. You can specify a different location via the environment variable *KAS_WORK_DIR*.

Command line usage

kas - setup tool for bitbake based project

```
usage: kas [-h] [--version] [-d] {build,shell} ...
```

Positional Arguments

cmd Possible choices: build, shell

sub command help

Named Arguments

--version show program's version number and exit

-d, --debug Enable debug logging

Default: False

Sub-commands:

build

Checks out all necessary repositories and builds using bitbake as specificed in the configuration file.

```
kas build [-h] [--target TARGET] [--task TASK] [--skip SKIP] config
```

Positional Arguments

config Config file

Named Arguments

--target Select target to build

--task Select which task should be executed

Default: "build"

--skip Skip build steps

Default: []

shell

Run a shell in the build environment.

```
kas shell [-h] [--target TARGET] [--skip SKIP] [-c COMMAND] config
```

Positional Arguments

config Config file

Named Arguments

--target Select target to build

Default: "core-image-minimal"

--skip Skip build steps

Default: []

-c, --command Run command

Default: ""

Environment variables

Environment variable name	Description
KAS_WORK_DIR	The path of the kas work directory, current work directory is the default.
KAS_REPO_REF_DIR	The path to the repository reference directory. Repositories in this directory are used as references when cloning. In order for kas to find those repositories, they have to be named correctly. Those names are derived from the repo url in the kas config. (E.g. url: "https://github.com/siemens/meta-iot2000.git" resolves to the name "github.com.siemens.meta-iot2000.git")
KAS_DISTRO KAS_MACHINE KAS_TARGET	This overwrites the respective setting in the configuration file.
SSH_PRIVATE_KEY	Path to the private key file, that should be added to an internal ssh-agent. This key cannot be password protected. This setting is useful for CI building server. On desktop machines, a ssh-agent running outside the kas environment is more useful.
DL_DIR SSTATE_DIR TMPDIR	Environment variables that are transfered to the bitbake environment.
http_proxy https_proxy no_proxy	This overwrites the proxy configuration in the configuration file.
SSH_AGENT_PID	SSH agent process id. Used for cloning over SSH.
SSH_AUTH_SOCK	SSH authenication socket. Used for cloning over SSH.
SHELL	The shell to start when using the *shell* plugin.
TERM	The terminal options used in the *shell* plugin.

Use Cases

1. Initial build/setup:

```
$ mkdir $PROJECT_DIR
$ cd $PROJECT_DIR
$ git clone $PROJECT_URL meta-project
$ kas build meta-project/kas-project.yml
```

2. Update/rebuild:

```
$ cd $PROJECT_DIR/meta-project
$ git pull
$ kas build kas-project.yml
```

Project Configuration

Two types of configuration file formats are supported.

For most purposes the static configuration should be used. In case this static configuration file does not provide enough options for customization, the dynamic configuration file format can be used.

Static project configuration

Currently JSON and YAML is supported as the base file format. Since YAML is arguable easier to read, this documentation focuses on the YAML format.

```
# Every file needs to contain a header, that provides kas with information
# about the context of this file.
header:
  # The `version` entry in the header describes for which kas version this
  # file was created. It is used by kas to figure out if it is compatible
  # with this file. Every version x.y.z should be compatible with
  # the configuration file version x.y. (x, y and z are numbers)
  version: "x.y"
# The machine as it is written into the `local.conf` of bitbake.
machine: qemu
# The distro name as it is written into the `local.conf` of bitbake.
distro: poky
repos:
  # This entry includes the repository where the config file is located
  # to the bblayers.conf:
  meta-custom:
  # Here we include a list of layers from the poky repository to the
  # bblayers.conf:
  poky:
    url: "https://git.yoctoproject.org/git/poky"
    refspec: 89e6c98d92887913cadf06b2adb97f26cde4849b
    layers:
      meta:
      meta-poky:
      meta-yocto-bsp:
```

A minimal input file consist out of the `header`, `machine`, `distro`, and `repos`.

Additionally, you can add `bblayers_conf_header` and `local_conf_header` which are strings that are added to the head of the respective files (`bblayers.conf` or `local.conf`):

```
bblayers_conf_header:
  meta-custom: |
    POKY_BBLAYERS_CONF_VERSION = "2"
    BBPATH = "${TOPDIR}"
    BBFILES ?= ""
local_conf_header:
  meta-custom: |
    PATCHRESOLVE = "noop"
    CONF_VERSION = "1"
    IMAGE_FSTYPES = "tar"
```

`meta-custom` in these examples should be a unique name (in project scope) for this configuration entries. We assume that your configuration file is part of a `meta-custom` repository/layer. This way its possible to overwrite or append entries in files that include this configuration by naming an entry the same (overwriting) or using a unused name (appending).

Including in-tree configuration files

Its currently possible to include kas configuration files from the same repository/layer like this:

```
header:
  version: "x.y"
  includes:
    - base.yml
    - bsp.yml
    - product.yml
```

The specified files are addressed relative to your current configuration file.

Including configuration files from other repos

Its also possible to include configuration files from other repos like this:

```
header:
  version: "x.y"
  includes:
    - repo: poky
      file: kas-poky.yml
    - repo: meta-bsp-collection
      file: hw1/kas-hw-bsp1.yml
    - repo: meta-custom
      file: products/product.yml
repos:
  meta-custom:
  meta-bsp-collection:
    url: "https://www.example.com/git/meta-bsp-collection"
    refspec: 3f786850e387550fdab836ed7e6dc881de23001b
    layers:
      # Additional to the layers that are added from this repository
      # in the hw1/kas-hw-bsp1.yml, we add here an additional bsp
      # meta layer:
      meta-custom-bsp:
  poky:
```

```
    url: "https://git.yoctoproject.org/git/poky"
    refspec: 89e6c98d92887913cadf06b2adb97f26cde4849b
    layers:
      # If `kas-poky.yml` adds the `meta-yocto-bsp` layer and we
      # do not want it in our bblayers for this project, we can
      # overwrite it by setting:
      meta-yocto-bsp: exclude
```

The files are addressed relative to the git repository path.

The include mechanism collects and merges the content from top to buttom and depth first. That means that settings in one include file are overwritten by settings in a latter include file and entries from the last include file can be overwritten by the current file. While merging all the dictionaries are merged recursive while preserving the order in which the entries are added to the dictionary. This means that `local_conf_header` entries are added to the `local.conf` file in the same order in which they are defined in the different include files. Note that the order of the configuration file entries is not preserved within one include file, because the parser creates normal unordered dictionaries.

Static configuration reference

- **`header`: dict [required]** The header of every kas configuration file. It contains information about context of the file.
  - **`version`: string [required]** Lets kas check if it is compatible with this file.
  - **`includes`: list [optional]** A list of configuration files this current file is based on. They are merged in order they are stated. So a latter one could overwrite settings from previous files. The current file can overwrite settings from every included file. An item in this list can have one of two types:
    - **item: string** The path to a kas configuration file, relative to the current file.
    - **item: dict** If files from other repositories should be included, choose this representation.
      - **`repo`: string [required]** The id of the repository where the file is located. The repo needs to be defined in the `repos` dictionary as `<repo-id>`.
      - **`file`: string [required]** The path to the file relative to the root of the repository.
- **`machine`: string [optional]** Contains the value of the `MACHINE` variable that is written into the `local.conf`. Can be overwritten by the `KAS_MACHINE` environment variable and defaults to `qemu`.
- **`distro`: string [optional]** Contains the value of the `DISTRO` variable that is written into the `local.conf`. Can be overwritten by the `KAS_DISTRO` environment variable and defaults to `poky`.
- **`target`: string [optional]** Contains the target to build by bitbake. Can be overwritten by the `KAS_TARGET` environment variable and defaults to `core-image-minimal`.
- **`repos`: dict [optional]** Contains the definitions of all available repos and layers.
  - **`<repo-id>`: dict [optional]** Contains the definition of a repository and the layers, that should be part of the build. If the value is `None`, the repository, where the current configuration file is located is defined as `<repo-id>` and added as a layer to the build.
    - **`name`: string [optional]** Defines under which name the repository is stored. If its missing the `<repo-id>` will be used.
    - **`url`: string [optional]** The url of the git repository. If this is missing, no git operations are performed.
    - **`refspec`: string [optional]** The refspec that should be used. Required if an `url` was specified.

        * **path: string [optional]** The path where the repository is stored. If the `url` and `path` is missing, the repository where the current configuration file is located is defined. If the `url` is missing and the path defined, this entry references the directory the path points to. If the `url` as well as the `path` is defined, the path is used to overwrite the checkout directory, that defaults to `kas_work_dir` + `repo.name`.
        * **layers: dict [optional]** Contains the layers from this repository that should be added to the `bblayers.conf`. If this is missing or `None` or and empty dictionary, the path to the repo itself is added as a layer.
            · **<layer-path>: enum [optional]** Adds the layer with `<layer-path>` that is relative to the repository root directory, to the `bblayers.conf` if the value of this entry is not in this list: `['disabled', 'excluded', 'n', 'no', '0', 'false']`. This way it is possible to overwrite the inclusion of a layer in latter loaded configuration files.

- **bblayers_conf_header: dict [optional]** This contains strings that should be added to the `bblayers.conf` before any layers are included.
  - **<bblayers-conf-id>: string [optional]** A string that is added to the `bblayers.conf`. The entry id (`<bblayers-conf-id>`) should be unique if lines should be added and can be the same from another included file, if this entry should be overwritten. The lines are added to `bblayers.conf` in the same order as they are included from the different configuration files.
- **local_conf_header: dict [optional]** This contains strings that should be added to the `local.conf`.
  - **<local-conf-id>: string [optional]** A string that is added to the `local.conf`. It operates in the same way as the `bblayers_conf_header` entry.
- **proxy_config: dict [optional]** Defines the proxy configuration bitbake should use. Every entry can be overwritten by the respective environment variables.
  - `http_proxy`: string [optional]
  - `https_proxy`: string [optional]
  - `no_proxy`: string [optional]

Dynamic project configuration

NOTE: Dynamic project configuration is experimental. The API may change or even be obsoleted in future versions. Please provide feedback if you consider it useful.

The dynamic project configuration is plain Python with following mandatory functions which need to be provided:

```
def get_machine(config):
    return 'qemu'


def get_distro(config):
    return 'poky'


def get_repos(target):
    repos = []

    repos.append(Repo(
        url='URL',
        refspec='REFSPEC'))

    repos.append(Repo(
```

```
        url='https://git.yoctoproject.org/git/poky',
        refspec='krogoth',
        layers=['meta', 'meta-poky', 'meta-yocto-bsp'])))

    return repos
```

Additionally, `get_bblayers_conf_header()`, `get_local_conf_header()` can be added.

```
def get_bblayers_conf_header():
    return """POKY_BBLAYERS_CONF_VERSION = "2"
BBPATH = "${TOPDIR}"
BBFILES ?= ""
"""


def get_local_conf_header():
    return """PATCHRESOLVE = "noop"
CONF_VERSION = "1"
IMAGE_FSTYPES = "tar"
"""
```

Furthermore, you can add pre and post hooks (`*_prepend`, `*_append`) for the exection steps in kas core, e.g.

```
def build_prepend(config):
    # disable distro check
    with open(config.build_dir + '/conf/sanity.conf', 'w') as f:
        f.write('\n')


def build_append(config):
    if 'CI' in os.environ:
        build_native_package(config)
        run_wic(config)
```

TODO: Document the complete configuration API.

CHAPTER 3

Developer Guide

Deploy for development

This project uses pip to manage the package. If you want to work on the project yourself you can create the necessary links via:

```
$ sudo pip3 install -e .
```

That will install a backlink /usr/bin/kas to this project. Now you are able to call it from anywhere.

Docker image build

Just run:

```
$ docker build -t <image_name> .
```

When you need a proxy to access the internet, add:

```
--build-arg http_proxy=<http_proxy> --build-arg https_proxy=<https_proxy>
```

to the call.

Community Resources

Project home:

- https://github.com/siemens/kas

Source code:

- https://github.com/siemens/kas.git

- git@github.com:siemens/kas.git

Documentation:

- https://kas.readthedocs.org

Mailing list:

- kas-devel@googlegroups.com
- Subscription:
  - kas-devel+subscribe@googlegroups.com
  - https://groups.google.com/forum/#!forum/kas-devel/join
- Archives
  - https://groups.google.com/forum/#!forum/kas-devel
  - https://www.mail-archive.com/kas-devel@googlegroups.com/

Class reference documentation

kas.kas Module

This module is the main entry point for kas, setup tool for bitbake based projects

kas.kas.**create_logger**()
Setup the logging environment

kas.kas.**interruption**()
Ignore SIGINT/SIGTERM in kas, let them be handled by our sub-processes

kas.kas.**kas**(*argv*)
The main entry point of kas.

kas.kas.**kas_get_argparser**()
Creates a argparser for kas with all plugins.

kas.kas.**main**()
The main function that operates as a wrapper around kas.

kas.libkas Module

This module contains the core implementation of kas.

class kas.libkas.**LogOutput**(*live*)
Handles the log output of executed applications

log_stderr(*line*)
This method is called when a line over stderr is received.

log_stdout(*line*)
This method is called when a line over stdout is received.

kas.libkas.**find_program**(*paths*, *name*)
Find a file within the paths array and returns its path.

kas.libkas.**get_build_environ**(*config*, *build_dir*)
Create the build environment variables.

kas.libkas.**kasplugin**(*plugin_class*)
A decorator that registeres kas plugins

kas.libkas.**repo_checkout**(*config*, *repo*)
Checks out the correct revision of the repo.

kas.libkas.**repos_fetch**(*config*, *repos*)
Fetches the list of repositories to the kas_work_dir.

kas.libkas.**run_cmd**(*cmd*, *cwd*, *env=None*, *fail=True*, *shell=False*, *liveupdate=True*)
Runs a command synchronously.

kas.libkas.**run_cmd_async**(*cmd*, *cwd*, *env=None*, *fail=True*, *shell=False*, *liveupdate=True*)
Run a command asynchronously.

kas.libkas.**ssh_add_key**(*env*, *key*)
Add ssh key to the ssh-agent

kas.libkas.**ssh_cleanup_agent**(*config*)
Removes the identities and stop the ssh-agent instance

kas.libkas.**ssh_no_host_key_check**(*_*)
Disables ssh host key check

kas.libkas.**ssh_setup_agent**(*config*, *envkeys=None*)
Starts the ssh-agent

kas.libcmds Module

This module contain common commands used by kas plugins.

class kas.libcmds.**CleanupSSHAgent**
Remove all the identities and stop the ssh-agent instance.

class kas.libcmds.**Command**
An abstract class that defines the interface of a command.

execute(*config*)
This method executes the command.

class kas.libcmds.**Macro**
Contains commands and provide method to run them.

add(*command*)
Appends commands to the command list.

run(*config*, *skip=None*)
Runs command from the command list respective to the configuration.

class kas.libcmds.**ReposCheckout**
Ensures that the right revision of each repo is check out.

class kas.libcmds.**ReposFetch**
Fetches repositories defined in the configuration

class kas.libcmds.**SetupDir**
Creates the build directory.

class kas.libcmds.**SetupEnviron**
Setups the kas environment.

class kas.libcmds.**SetupHome**
Setups the home directory of kas.

class kas.libcmds.**SetupProxy**
Setups proxy configuration in the kas environment.

class kas.libcmds.**SetupSSHAgent**
Setup the ssh agent configuration.

class kas.libcmds.**WriteConfig**
Writes bitbake configuration files into the build directory.

kas.build Module

The build plugin for kas.

class kas.build.**BuildCommand**(*task*)
Implement the bitbake build step.

execute(*config*)
Executes the bitbake build command.

kas.shell Module

This module contains a kas plugin that opens a shell within the kas environment

class kas.shell.**ShellCommand**(*cmd*)
This class implements the command that starts a shell.

kas.config Module

This module contains the implementation of the kas configuration.

class kas.config.**Config**
This is an abstract class, that defines the interface of the kas configuration.

build_dir
The path of the build directory.

get_bblayers_conf_header()
Returns the bblayers.conf header

get_bitbake_target()
Return the bitbake target

get_distro()
Returns the distro

get_gitlabci_config()
Returns the GitlabCI configuration

get_hook(*fname*)
Returns a function that is executed instead of the command or None.

get_local_conf_header()
Returns the local.conf header

get_machine()
Returns the machine

get_proxy_config()
Returns the proxy settings

get_repo_ref_dir()
The path to the directory that contains the repository references.

get_repos()
Returns the list of repos.

kas_work_dir
The path to the kas work directory.

post_hook(*fname*)
Returs a function that is executed after every command or None.

pre_hook(*fname*)
Returns a function that is executed before every command or None.

setup_environ()
Sets the environment variables for process that are started by kas.

class kas.config.**ConfigPython**(*filename*, *target*)
Implementation of a configuration that uses a Python script.

create_config(*target*)
Sets the configuration for *target*

get_bblayers_conf_header()
Returns the bblayers.conf header

get_bitbake_target()
Return the bitbake target

get_distro()
Returns the distro

get_gitlabci_config()
Returns the GitlabCI configuration

get_local_conf_header()
Returns the local.conf header

get_machine()
Returns the machine

get_target()
Returns the target

class kas.config.**ConfigStatic**(*filename*, *target*)
Implements the static kas configuration based on config files.

get_repo_dict()
Returns a dictionary containing the repositories with their name (as it is defined in the config file) as key and the *Repo* instances as value.

get_repos()
Returns the list of repos.

kas.config.**get_distro_id**()
Wrapper around platform.dist to simulate distro.id platform.dist is deprecated and will be removed in python 3.7 Use the 'distro' package instead.

kas.config.**load_config**(*filename*, *target*)
Return configuration generated from *filename*.

kas.repos Module

This module contains the Repo class.

class kas.repos.**Repo**(*url*, *path*, *refspec=None*, *layers=None*)
Represents a repository in the kas configuration.

disable_git_operations()
Disabled all git operation for this repository.

kas.includehandler Module

This module implements how includes of configuration files are handled in kas.

class kas.includehandler.**GlobalIncludes**(*top_file*)
Implements a handler where every configuration file should contain a dictionary as the base type with and 'includes' key containing a list of includes.

The includes can be specified in two ways, as a string containg the relative path from the current file or as a dictionary. The dictionary should have a 'file' key, containing the relative path to the include file and optionally a 'repo' key, containing the key of the repository. If the 'repo' key is missing the value of the 'file' key is threated the same as if just a string was defined, meaning the path is relative to the current config file otherwise its relative to the repository path.

The includes are read and merged depth first from top to buttom.

exception kas.includehandler.**IncludeException**
Class for exceptions that appear in the include mechanism.

class kas.includehandler.**IncludeHandler**(*top_file*)
Abstract class that defines the interface of an include handler.

get_config(*repos=None*)

Parameters: repos – A dictionary that maps repo name to directory path

Returns:

(config, repos) config – A dictionary containing the configuration repos – A list of missing repo names that are needed to create a complete configuration

exception kas.includehandler.**LoadConfigException**
Class for exceptions that appear while loading the configuration file.

kas.includehandler.**load_config**(*filename*)
Load the configuration file and test if version is supported.

CHAPTER 4

Indices and tables

- genindex
- modindex
- search

Python Module Index

k

Index

M

P

R

S

W