kas Documentation

Release 2.0

Daniel Wagner, Jan Kiszka, Claudius Heine

May 27, 2020

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

2.1 Dependencies & installation

This project depends on

- Python 3
- distro Python 3 package
- jsonschema 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 instead of the asyncio.

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

```
$ sudo pip3 install .
```

2.2 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 kasproject/kas:<version> sh` or bind-mount the project into the container. See https://hub.docker.com/r/kasproject for all available images.
- 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*.

2.2.1 Command line usage

2.2.2 Environment variables

Environment variables	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 in a specific way. The repo URLs are translated like this: "https://github.com/siemens/meta-iot2000.git" resolves to the name "github.com.siemens.meta-iot2000.git".

`KAS_DISTRO`
`KAS_MACHINE`
`KAS_TARGET`
`KAS_TASK`
	This overwrites the respective setting in the configuration file.

`KAS_PREMIRRORS`	Specifies alternatives for repo URLs. Just like bitbake `PREMIRRORS`, this variable consists of new-line separated entries. Each entry defines a regular expression to match a URL and, space-separated, its replacement. E.g.: "https://.*.somehost.io/ https://localmirror.net/"
`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 build servers. On desktop machines, an ssh-agent running outside the kas environment is more useful.
`SSH_AUTH_SOCK`	SSH authentication socket. Used for cloning over SSH (alternative to `SSH_PRIVATE_KEY`).

`DL_DIR`
`SSTATE_DIR`
`TMPDIR`
	Environment variables that are transferred to the bitbake environment.

`http_proxy`
`https_proxy`
`ftp_proxy`
`no_proxy`
	This overwrites the proxy configuration in the configuration file.

`GIT_PROXY_COMMAND` `NO_PROXY`	Set proxy for native git fetches. `NO_PROXY` is evaluated by OpenEmbedded's oe-git-proxy script.
`SHELL`	The shell to start when using the *shell* plugin.
`TERM`	The terminal options used in the *shell* plugin.

2.3 Use Cases

1. Initial build/setup:

```
$ mkdir $PROJECT_DIR
$ cd $PROJECT_DIR
```

(continues on next page)

(continued from previous page)

```
$ 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
```

2.4 Project Configuration

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

```yaml
# 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 configuration
  # format version this file was created for. It is used by kas to figure
  # out if it is compatible with this file. The version is an integer that
  # is increased on every format change.
  version: x
# 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 consists 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`):

```yaml
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 an unused name (appending).

2.4.1 Including in-tree configuration files

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

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

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

2.4.2 Including configuration files from other repos

It's also possible to include configuration files from other repos like this:

```
header:
  version: x
  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 recursively 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.

2.4.3 Including configuration files via the command line

When specifying the kas configuration file on the command line, additional configurations can be included ad-hoc:

```
$ kas build kas-base.yml:debug-image.yml:board.yml
```

This is equivalent to static inclusion from some kas-combined.yml like this:

```
header:
  version: x
  includes:
    - kas-base.yml
    - debug.image.yml
    - board.yml
```

Command line inclusion allows to create configurations on-demand, without the need to write a kas configuration file for each possible combination.

Note that all configuration files combined via the command line either have to come from the same repository or have to live outside of any versioning control. kas will refuse any other combination in order to avoid complications and configuration flaws that can easily emerge from them.

2.4.4 Configuration reference

- **`header`: dict [required]** The header of every kas configuration file. It contains information about the context of the file.
  - **`version`: integer [required]** Lets kas check if it is compatible with this file. See the *configuration format changelog* for the format history and the latest available version.
  - **`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] or list [optional]** Contains the target or a list of targets to build by bitbake. Can be overwritten by the `KAS_TARGET` environment variable and defaults to `core-image-minimal`. Space is used as a delimiter if multiple targets should be specified via the environment variable.
- **`env`: dict [optional]** Contains environment variable names with the default values. These variables are made available to bitbake via `BB_ENV_EXTRAWHITE` and can be overwritten by the variables of the environment in which kas is started.

- **task: string [optional]** Contains the task to build by bitbake. Can be overwritten by the KAS_TASK environment variable and defaults to build.
- **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 repository. If this is missing, no version control operations are performed.
    - **type: string [optional]** The type of version control repository. The default value is git and hg is also supported.
    - **refspec: string [optional]** The refspec that should be used. If url was specified but no refspec the revision you get depends on the defaults of the version control system used.
    - **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. In case of a relative path name kas_work_dir is prepended.
    - **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.
    - **patches: dict [optional]** Contains the patches that should be applied to this repo before it is used.
      - **<patches-id>: dict [optional]** One entry in patches with its specific and unique id. All available patch entries are applied in the order of their sorted <patches-id>.
      - **repo: string [required]** The identifier of the repo where the path of this entry is relative to.
      - **path: string [required]** The path to one patch file or a quilt formatted patchset directory.
- **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]

CHAPTER 3

Developer Guide

3.1 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:

```
$ pip3 install --user -e .
```

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

3.2 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> --build-arg␣
↪ftp_proxy=<ftp_proxy> --build-arg no_proxy=<no_proxy>
```

to the call.

3.3 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/

3.4 Class reference documentation

3.4.1 `kas.kas` Module

3.4.2 `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 is received over stderr.

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

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

kas.libkas.**get_build_environ**()
Creates the build environment variables.

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

kas.libkas.**repos_apply_patches**(*repos*)
Applies the patches to the repositories.

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

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

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

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

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

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

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

3.4.3 kas.libcmds Module

3.4.4 kas.build Module

3.4.5 kas.shell Module

3.4.6 kas.config Module

3.4.7 kas.repos Module

This module contains the Repo class.

class kas.repos.**GitRepo**(*url*, *path*, *refspec*, *layers*, *patches*, *disable_operations*)
Provides the git functionality for a Repo.

class kas.repos.**MercurialRepo**(*url*, *path*, *refspec*, *layers*, *patches*, *disable_operations*)
Provides the hg functionality for a Repo.

class kas.repos.**Repo**(*url*, *path*, *refspec*, *layers*, *patches*, *disable_operations*)
Represents a repository in the kas configuration.

static factory(*name*, *repo_config*, *repo_fallback_path*)
Returns a Repo instance depending on params.

static get_root_path(*path*, *fallback=True*)
Checks if path is under version control and returns its root path.

class kas.repos.**RepoImpl**(*url*, *path*, *refspec*, *layers*, *patches*, *disable_operations*)
Provides a generic implementation for a Repo.

apply_patches_async()
Applies patches to a repository asynchronously.

checkout()
Checks out the correct revision of the repo.

fetch_async()
Starts asynchronous repository fetch.

3.4.8 kas.includehandler Module

CHAPTER 4

Configuration Format Changes

4.1 Version 1 (Alias '0.10')

4.1.1 Added

- Include mechanism
- Version check

4.2 Version 2

4.2.1 Changed

- Configuration file versions are now integers

4.2.2 Fixed

- Including files from repos that are not defined in the current file

4.3 Version 3

4.3.1 Added

- `Task` key that allows to specify which task to run (`bitbake -c`)

4.4 Version 4

4.4.1 Added

- `Target` key now allows to be a list of target names

4.5 Version 5

4.5.1 Changed behavior

- Using `multiconfig:*` targets adds appropriate `BBMULTICONFIG` entries to the `local.conf` automatically.

4.6 Version 6

4.6.1 Added

- `env` key now allows to pass custom environment variables to the bitbake build process.

4.7 Version 7

4.7.1 Added

- `type` property to `repos` to be able to express which version control system to use.

4.8 Version 8

4.8.1 Added

- `patches` property to `repos` to be able to apply additional patches to the repo.

CHAPTER 5

Indices and tables

- genindex
- modindex
- search

Python Module Index

k

Index