Contents

Embedded Development Environment (EDE) 3.x.x - User Manual

Przemysław Guza, Grzegorz Borowiak

March 10, 2022

Chapter 1

EDE user manual

1.1 What is EDE

The ADVA-OS Embedded Development Environment, or EDE for short, is a cross-compilation environment specifically designed to support the AOS Embedded Development community.

Currently supported are the following target platforms:

- x86_64 (amd64), 64-bit
- x86 (i686), 32-bit
- powerpc-e500v2, 32-bit
- powerpc-e5500, 32-bit [1]
- powerpc64-e5500, 64-bit
- armv7a (arm), 32-bit
- armv5tel, 32-bit
- aarch64 (arm64), 64-bit

The essential ingredients of EDE are cross-toolchains for target architectures and sets of standard libraries and executables, already cross-compiled for these target architectures. rsion

1.2 Request a new package to the EDE

All requests for changes/packages to be added to the ***EDE must be*** submitted to ***Paprykarz Team (EDE)*** in Polarion. Also should be specified where the package should be installed. (tools, targets)

The *highest priority* is set on packages in embedded toolchains and packages dependencies in that and according to this requirement, the *EDE Tools* are customized for embedded toolchains, because many cross-compiled packages requires native tools in *special* versions.

Typical dependencies graph for embedded toolchains:

- dependencies between embedded packages (names and versions)

  for runtime

  for install

  for compilation

- dependencies between embedded packages and native tools (names and versions)

  for pre-compile process

  for compilation

  requires proper versions and compilation flags in native and cross toolchain

[1] cross-binutils and cross-gcc only

The *EDE native tools* are prepared mainly for embedded toolchains and many packages versions in *native tools* are limited by *cross-compiled* packages. In this case is no possible install all native tools in EDE from all currently available versions.

1.3 Installation Guide

- create *EDE* path Choose a location on your host *PC* where you would like to install *EDE*. Here we assume `/opt/adva/ede`.
  `sudo mkdir -p /opt/adva/ede`.
- *EDE* host requirements - *rsync tool*

```
sudo apt-get install rsync (Debian, Ubuntu)
sudo yum install rsync     (Red Hat, Centos)
sudo pacman -S rsync       (Arch Linux)
sudo emerge rsync          (Gentoo)
```

- download the *EDE* from the *EDE Distribution Server*
  EDE mirrors: [2]
  - hostname: *GDN-S-SYS-EDE-1.advaoptical.com*
  - hostname: *GDN-S-SYS-EDE-2.advaoptical.com*

```
ede-2.x.x
sudo rsync -avz --delete rsync://GDN-S-SYS-EDE-1.advaoptical.com/get-ede/ <ede path>
sudo rsync -avz --delete rsync://GDN-S-SYS-EDE-2.advaoptical.com/get-ede/ <ede path>
ede-3.x.x
sudo rsync -ai rsync://GDN-S-SYS-EDE-1.advaoptical.com/ede/base/<ede tag>/. <ede path>/.
sudo rsync -ai rsync://GDN-S-SYS-EDE-2.advaoptical.com/ede/base/<ede tag>/. <ede path>/.
```

Example:

```
ede-2.x.x
sudo mkdir -p /opt/adva/ede
sudo rsync -avz --delete rsync://GDN-S-SYS-EDE-1.advaoptical.com/get-ede/ /opt/adva/ede
sudo rsync -avz --delete rsync://GDN-S-SYS-EDE-2.advaoptical.com/get-ede/ /opt/adva/ede
ede-3.x.x
sudo mkdir -p /opt/adva/ede-3
sudo rsync -ai rsync://GDN-S-SYS-EDE-1.advaoptical.com/ede/base/3.6.3/. /opt/adva/ede-3/.
sudo rsync -ai rsync://GDN-S-SYS-EDE-2.advaoptical.com/ede/base/3.6.3/. /opt/adva/ede-3/.
```

1.3.1 Install EDE by dedicated script.

The EDE can be installed by script which is available in:
`https://polarion.advaoptical.com/polarion/wiki/bin/download/project/AOS_Dev/page/Developers%20Portal/Embedded%20development%20environment/ede_install.py`

The installer requires packages on host: `rsync, python3-dialog`.
`(sudo apt-get install python3-dialog rsync)`

run with sudo:

```
sudo ./ede_install.py
```

- run EDE
  As root

[2] Currently is preferred use EDE server names instead of ip addresses. The servers names are visible in ADVA network only.

```
sudo /opt/adva/ede/rehome/reho -r
```

As user

```
sudo /opt/adva/ede/rehome/reho -u $USER
```

- Mount external disks

  ```
  sudo /opt/adva/ede/rehome/reho -u $USER -e /<host path>/:/<in EDE path>
  ```

  Example:

  ```
  sudo /opt/adva/ede/rehome/reho -u $USER-e /mnt/gitRepos/:/mnt/gitRepos
  ```

  The mount point in EDE should be created before use it.

- EDE rsync command *(run in EDE)*

  ```
  ede_rsync <ede version>
  ```

  Example:

  ```
  ede_rsync 3.6.3
  ede_rsync 3.6.3 --yes (with confirming)
  ```

- EDE set up modules *(run in EDE)*

  ```
  ede_modules_config
  ```

  Example modules list:

  ```
  [ ] *arch--aarch64-linux-gnu
  [ ] *arch--armv7a-hardfloat-linux-gnueabi
  [ ] *arch--i686-vm-linux-gnu
  [ ] *arch--powerpc-e500v2-linux-gnu
  [ ] *arch--powerpc-e5500-linux-gnu
  [ ] *arch--powerpc-softfloat-linux-gnu
  [ ] *arch--powerpc64-e5500-linux-gnu
  [ ] *arch--x86_64-vm-linux-gnu
  [ ] *portage
  ...
  ```

1.3.2 Download product sources

The source code for all cross compiled packages are available on EDE servers, and it can be downloaded by command:

```
sudo rsync -ai rsync://GDN-S-SYS-EDE-1.advaoptical.com/ede/src/3.6.8/. /opt/adva/ede-src/.
sudo rsync -ai rsync://GDN-S-SYS-EDE-2.advaoptical.com/ede/src/3.6.8/. /opt/adva/ede-src/.
```

1.4 Documentation sources

The Documentation is stored in *aos-ne-os* repository. [3]
The main UNIX rule: KISS. (keep it stupid simple) [4]

[3] url: ssh://*user_name*@muc-gerrit.rd.advaoptical.com:29418/aos-ne-os

[4] url: https://en.wikipedia.org/wiki/KISS_principle

1.4.1 Building documentation

Go to directory: aos-ne-os/Documentation/ede-3.x.x/
Run two times: (second for *TOC*) [5]

```
pdflatex manual.tex
```

[6] Result:
`manual.pdf` [7] Build documenetation step generates:

- */Documentation/manual.pdf*
- */etc/ede/EDE-packages.pdf/txt*

In each released *EDE* the *Release Notes* are stored in: */etc/ede/ede_history*. It consist of all packages changes in tools and embedded (targets).

1.5 Release Notes

EDE *release notes/history/changes* are available in each *EDE* version in file: */etc/ede/ede_history*.

Example:

```
less /etc/ede/ede_history
ede-3.3.2  /

--------------------------
ede-3.3.2
--------------------------

* native package changes since ede-3.3.1

    * new

        app-shells/bash-completion-2.9-r1
        app-shells/gentoo-bashcomp-20180302
        sys-apps/miscfiles-1.5-r3

    * discontinued

        dev-util/artifactory-bin-6.3.3

--------------------------
ede-3.3.1
--------------------------

* native package changes since ede-3.3.0

    * new

        dev-util/artifactory-bin-6.3.3
        dev-libs/libnl-3.4.0

    * upgraded/downgraded

        sys-fs/populatefs-mod-0.9a (from 0.9)
```

[5] table of contents

[6] LaTeX package is required for this operation.

[7] This document is generated on *Jenkins* in *EDE* releasing flow in last step. It is something like a build stamp.

```
* cross package changes since ede-3.3.0

    * upgraded/downgraded

        eos/cs_init-55 (from 54)
        net-misc/curl-7.58.0 (from 7.66.0)
        dev-libs/farmhash-0.20190513 (from 0.20180817)
        eos/ir_init-55 (from 54)
        net-libs/libssh2-1.8.0-r1 (from 1.9.0-r1)

--------------------------
...
```

1.6 EDE Release information

The information about *packages versions* are available in text files in *EDE*:

```
/etc/ede/EDE-packages.txt
or
/etc/ede/EDE-packages.pdf
```

The file consist of sections:

- *EDE* release date
- *Gentoo* stage and portage date
- Packages in embedded targets

  it contains all available cross-compiled packages in toolchains.
- Canon CC [8]

  it contains canon packages list. It is used also for BOM[9] in Black Duck.[10]
- Canon EC *(it expands CC)*

  it expands canon packages list to *EC*
- EDE tools - packages

  it contains packages versions available in *EDE Tools.*
- EDE security level content

  it contains security level information, while EDE product was released.

1.7 Packages dependencies

The information about *packages dependencies* are available in text files in:

```
/etc/ede/powerpc64-e5500-linux-gnu.txt
/etc/ede/x86_64-vm-linux-gnu.txt
```

[11]

[8] Canon packages list on CC*(F8)*)

[9] Bill Of Material

[10] Black Duck Software Composition Analysis

[11] The *x86_64* has more packages than other embedded toolchains.

1.8 Checking possible security issues

The information about *security updates* can be shown by command: [12]

```
check_security_on_ede.sh
```

Example:

```
check_security_on_ede.sh
ede_rsync (Y/n)
...
---- >amd64<----
[A] means this GLSA was marked as applied (injected),
[U] means the system is not affected and
[N] indicates that the system might be affected.

202005-09 [N] Python: Denial of Service ( dev-lang/python )
202007-54 [N] rsync: Multiple vulnerabilities ( net-misc/rsync )
202008-01 [N] Python: Multiple vulnerabilities ( dev-lang/python )
202012-06 [N] Linux-PAM: Authentication bypass ( sys-libs/pam )
202101-18 [N] Python: Multiple vulnerabilities ( dev-lang/python )
202101-20 [N] glibc: Multiple vulnerabilities ( sys-libs/glibc )
---- >ppc64<----
[A] means this GLSA was marked as applied (injected),
[U] means the system is not affected and
[N] indicates that the system might be affected.

202005-09 [N] Python: Denial of Service ( dev-lang/python )
202007-54 [N] rsync: Multiple vulnerabilities ( net-misc/rsync )
202008-01 [N] Python: Multiple vulnerabilities ( dev-lang/python )
202012-06 [N] Linux-PAM: Authentication bypass ( sys-libs/pam )
202101-18 [N] Python: Multiple vulnerabilities ( dev-lang/python )
202101-20 [N] glibc: Multiple vulnerabilities ( sys-libs/glibc )
ede_rsync (Y/n)
...
```

[12] This command is available since *EDE* version 3.6.5 *(ede-3.6.5).*

Chapter 2

EDE tools

2.1 EDE specific commands

2.1.1 ede_rsync

The command `ede_rsync` syncing currently used *EDE tag* with *EDE server*. Any local changes will be removed after that. Possible options: [1]

```
ede_rsync -h
usage: ede_rsync [-h] [-n] [-y] [-a] [-d]

optional arguments:
  -h, --help     show this help message and exit
  -n, --dry-run  show what would be rsynced, do not rsync
  -y, --yes      just rsync, without asking
  -a, --ask      show what would be rsynced, ask whether rsync or not
  -d, --verbose  add verbosity
  -s, --ha       checks EDE mirror servers and the best server is set in configuration
                 file /etc/ede/sync/mirrors.auto
```

Example:

```
ede_rsync --yes
```

2.1.2 ede_switch

The command `ede_switch` switches *EDE* to required tag.

```
ede_switch -h
usage: ede_switch [-h] [-n] [-y] [-a] LABEL

positional arguments:
  LABEL          switch to EDE version specified by LABEL

optional arguments:
  -h, --help     show this help message and exit
  -n, --dry-run  show what would be rsynced, do not rsync
  -y, --yes      just rsync, without asking
  -a, --ask      show what would be rsynced, ask whether rsync or not
  -s, --ha       checks EDE mirror servers and the best server is set in configuration
                 file /etc/ede/sync/mirrors.auto
```

Example:

```
ede_switch 3.1.14 --yes
```

[1] `-s, --ha option is available since ede-3.8.5 version`

2.1.3 ede_modules_config

The command `ede_modules_config` sets required *EDE modules*.

```
Select modules:
[ ] *arch--aarch64-linux-gnu
[ ] *arch--armv7a-hardfloat-linux-gnueabi
[*] *arch--i686-vm-linux-gnu
[ ] *arch--powerpc-e500v2-linux-gnu
[ ] *arch--powerpc-e5500-linux-gnu
[*] *arch--powerpc64-e5500-linux-gnu
[ ] *arch--x86_64-nfv-linux-gnu
[*] *arch--x86_64-vm-linux-gnu
[*] *portage
...
    <  OK  >        <Cancel>
```

2.1.4 ede_verify

The command `ede_verify` verifying md5sum of all files in *EDE*. [2]
Example:

```
    ede_verify
    metadata...
    symlinks...
    contents...
    OK
```

More information is available in help.

```
    ede_verify --help
```

2.1.5 ede_ha

[3] This command checks *EDE* mirror servers and the best server is set in configuration file */etc/ede/sync/mirrors.auto*. Contents of this file is replaced in this case. From this moment, the command *ede_rsync* uses the best response *EDE* mirror.

```
ede_ha
[ start ] EDE check mirrors
    INFO: ip -  10.143.218.3 latency -  0.035
    INFO: The file - /etc/ede/sync/mirrors.auto is written.
[  ok   ] EDE check mirrors
```

2.1.6 *red list* config

Any local changes are cleaned after use command *ede_rsync*. For preventing this operation is possible protect a local changed files by *red list* mechanism.

- create file *redlist.user*
  In *ede-2.x.x*: `/etc/ede/rsync/redlist.user`
  In *ede-3.x.x*: `/etc/ede/sync/redlist.user`

- *redlist.user* content, example:

  ```
  /opt/jdk/x64/jdk1.8.0_121
  /opt/jdk/i586/jdk1.8.0_121
  ```

  From this moment the command *ede_rsync* not removing this dirs and files.

[2] This command is available since ede-3.6.0.
[3] High Availability

2.1.7 adva_deploy

The command `adva_deploy` generates rootfs or disk image for embedded systems.

- *(-d) option*
  This parameter is mandatory. It specifies *product name.*

  ```
  adva_deploy -h

  ERROR
  specify -d <product>
  ```

  Example:

  -d vm64

  -d Fred

- *(-i) or (-p) option without parameter*
  This option is mandatory. *(-i)* generates disk image, *(-p) generates archiv.*

- pre-sets *(-s) option*

  onedisk, creates one disk with many partitions. Without this option, *adva_deploy* generates one disk img file per partition.

  vga, creates disk image with configured *VGA* display.

  syslinux or efi, [4] creates disk image with bootloader *syslinux/extlinux or efi.*
  Example:

  ```
  -s onedisk -s syslinux -s vga
  ```

product configuration file

Typical product configuration file: [5]

```
specimen:
    ifs:
        eth0:
            ip: 192.168.122.208/24
    hostname: Fred-jenkins

volumes:
    adva:
        size: 4096
        type: ext4
        writable: 1
    main:
        size: 1024
        type: ext4
        writable: 1
    rwd:
        cskey: main
        size: 1024
        type: ext4
        writable: 1
    rwda:
        cskey: adva
```

[4] Option for vm64 product only.

[5] (for example *vm64/Fred*)

```
        size: 1024
        type: ext4
        writable: 1
    staging:
        size: 2356
        type: ext4
        writable: 1
        quantity: 1

container:
    banks: 2

vm.disk_format: vdi
proxy:
    path: /output/products/proxy_for_vm64
    stay: 1

install:
    path: /output/products/vm64
    overwrite: 1

packages:
    - add valgrind
    - add libssh
    - add libxslt
    - add sys-apps/acl
    - add dev-libs/libunistring
    - add net-dns/libidn2
    - add dev-libs/libpcre
    - add dev-util/perf

fsmods_local:
    - lxc
    - udev
    - otherPackages
```

Product configuration file sections (description):

- *specimen*

  ```
  specimen:
      ifs:
          eth0:
              ip: 192.168.122.208/24
      hostname: Fred-jenkins
  ```

  This section consist of network configuration, hostname etc. Possible options *(more network interfaces, dns, ntp)*:

  ```
  specimen:
      ifs:
         eth0:
              ip: 192.168.1.50/8
         eth1:
              ip: 192.168.1.51/8
      dns:
      - 172.27.1.100
  ```

```
    - 172.27.1.15
    ntp:
    - 172.25.1.141
    hostname: f7ncuII
```

- *volumes*

```
volumes:
    adva:
        size: 4096
        type: ext4
        writable: 1
    main:
        size: 1024
        type: ext4
        writable: 1
    rwd:
        cskey: main
        size: 1024
        type: ext4
        writable: 1
    rwda:
        cskey: adva
        size: 1024
        type: ext4
        writable: 1
    staging:
        size: 2356
        type: ext4
        writable: 1
        quantity: 1
```

This is a partition layout configuration.

volumes, format types

```
    main:
        size: 1024
        type: ext4/ext3/ext2/9p
        writable: 1
```

[6]

volumes, partition numbers definition

```
volumes:
    main:
        size: 1024
        type: ext4
        writable: 1
        devs: 5 7
    boot:
        devs: 6 8
```

Result:

[6] 9p should be used without a *pre-sets* (adva_deploy with -s options).

```
cs_tab content:
true    /dev/sda1    cs         ext4    ro
true    /dev/sda2    modules    ext4    ro
true    /dev/sda3    adva       ext4    ro
true    /dev/sda4    rwd        ext4    rw
true    /dev/sda5    main       ext4    rw
true    /dev/sda6    boot       vfat    ro
true    /dev/sda7    rwda       ext4    rw
```

volumes, remove cs partition from a list

```
volumes:
    ~cs: {}
```

- *container*

```
container:
    banks: 2
```

Set up a redundant partitions.
In this section also is possible set *disk type* or *disk size.*

```
container:
    type: gpt/mbr
    size: 4096
```

- *vm.disk_format*

```
vm.disk_format: vdi
```

Disk type image definition. It is currently used for vm64/Fred product.

- *proxy*

```
proxy:
    path: /output/products/proxy_for_vm64
    stay: 1
```

Path definition, removing or not temporary files (stay) which are required for generating disk image.

```
proxy:
    path: /output/products/proxy_for_vm64
    stay: 1
    only: 1
    result: /output/products/rootfs
    overwrite: 1
```

For preparing *rootfs* only, the variables: *"result"* and *"only"* are required. In this case *adva_deploy* generates *rootfs* only.

- *install*

```
install:
    path: /output/products/vm64
    overwrite: 1
```

Results path definition.
Fast test all available packages on target are possible by set param *(everything)*:

```
install:
    path: /output/products/vm64
    overwrite: 1
    everything: 1
```

- *packages*

  ```
  packages:
      - add valgrind
      - add libssh
      - add libxslt
      - add sys-apps/acl
      - add dev-libs/libunistring
      - add net-dns/libidn2
      - add dev-libs/libpcre
      - add dev-util/perf
  ```

  Additional packages which are not available in canon list but are installed on (embedded) target.

  ```
  packages:
      - remove valgrind
  ```

  Removing packages, for example package valgrind is removed from *canon* list before generate disk image.

- *fsmods_local*

  ```
  fsmods_local:
      - lxc
      - udev
      - otherPackages
  ```

  Local packages, configuration files, which are defined in product path. Example:

  ```
  ls -l /targ/prod/Fred/fsmods/
  total 20
  drwxrwxr-x 4 1000 1000 4096 Sep 25 11:22 lxc
  drwxr-xr-x 7 root root 4096 Nov  8 09:51 otherPackages
  drwxrwxr-x 3 1000 1000 4096 Sep 25 11:22 udev
  ```

  When the product is mounted to mount point: */targ/prod-devel/**product name***, the *fsmods* directory should be created in path:
  */targ/prod-devel/**product name**/copy/fsmods* and before use *adva_deploy* command, should be called command:
  prod_copy_sync ***product name***.

  Example:

  ```
  ls -l /targ/prod-devel/vm64/copy/fsmods/
  drwxrwxr-x 3 1000 1000 4096 Sep 25 11:22 etc

  prod_copy_sync vm64

  ls -l /targ/prod/vm64/fsmods/
  drwxrwxr-x 3 1000 1000 4096 Sep 25 11:22 etc
  ```

- *fsmods_local* - use scripts

  Also it's possible use scripts included in *fsmods* which are called by *deploy_hooks.py* while runing *adva_deploy.*

  Example: *(removing file)* [7]

  Creating required directory and file:

```
mkdir -p /targ/prod-devel/vm64/copy/fsmods/files_remove/"#scripts"
touch 1
chmod 755 1
```

  Script: *(1)*

```
    #!/bin/sh -e
    rm "$1"/usr/bin/less
    echo OK
```

  The file */usr/bin/less* was removed while *adva_deploy* running.

- *fsmod* - use scripts

  Also it's possible use scripts included in product directory in *fsmod* which are called by *deploy_hooks.py* while runing *adva_deploy.*

  Example: Creating required directory and file:

```
mkdir -p /targ/prod-devel/vm64/copy/fsmod/"#scripts"
touch 1
chmod 755 1
```

  Script: *(1)*

```
    #!/bin/sh -e
    chmod 000 "$1"/sbin/reboot
    chmod 000 "$1"/sbin/halt
    chmod 000 "$1"/sbin/shutdown
    rm "$1"/sbin/poweroff
    echo OK
```

  Script: *(2) - create fsmods local: customize*

  Create file: */targ/prod-devel/vm64/copy/fsmods/customize/#'scripts'/1* [8]

  Set permission: *chmod 755 /targ/prod-devel/vm64/copy/fsmods/customize/#'scripts'/1.*

```
#!/bin/sh -e
# create dir on generated rootfs
mkdir -p $1/customize
# create file (for example)
touch $1/customize/info.txt
# create next file with content from echo command (for example)
echo "Generated content in script." > $1/customize/Readme.txt
```

  Add section to product configuration file:

```
fsmods_local:
    - customize
```

[7] Example for vm64 product.

[8] This script will be copied by *adva_deploy* to: */targ/prod/vm64/fsmods/customize/#scripts/1'*

Generate rootfs or disk img

```
adva_deploy -i -d vm64 -f /targ/prod-devel/vm64/conf-vm64-rootfs
```

Results:

```
# ls -l  /output/products/install-vm64/rootfs/customize/
total 4
-rw-r--r-- 1 root root 29 Feb 22 07:54 Readme.txt
-rw-r--r-- 1 root root  0 Feb 22 07:54 info.txt
```

The script ***1*** will be called by *adva_deploy*.

2.1.8 createPackage.py

The script `createPackage.py` creates archiv with required package.

```
createPackage.py -h
Please go to required rootfs and type package name as a param.
Example:
        cd /targ/arch/x86_64-vm-linux-gnu/modes/eos
        createPackage.py "valgrind"
```

Example with result:

```
createPackage.py "valgrind"
Package name: (parameter)       valgrind
        Create package with dependencies: (Y/n)
n
         dependencies=False
EDE version:                    ede-3.0.0-pre46

Package name: (full)            dev-util/valgrind-3.14.0
Toolchain name:                 x86_64-vm-linux-gnu
         Depend packages:
                        app-portage/elt-patches-20170815
                        sys-devel/gettext-0.18.1.1-r3


                        sys-devel/automake-1.16.1:1.16
                        sys-devel/automake-1.15.1:1.15

                        sys-devel/autoconf-2.69
                        sys-devel/libtool-2.4
Package name:   ede-3.0.0-pre46--x86_64-vm-linux-gnu--dev-util--valgrind-3.14.0.tar.gz
Result: (in current directory)
        ede-3.0.0-pre46--x86_64-vm-linux-gnu--dev-util--valgrind-3.14.0.tar.gz


ls ede-3.0.0-pre46--x86_64-vm-linux-gnu--dev-util--valgrind-3.14.0.tar.gz
ede-3.0.0-pre46--x86_64-vm-linux-gnu--dev-util--valgrind-3.14.0.tar.gz
```

Solution for scripts

Example with confirming:

```
cd /usr/powerpc64-e5500-linux-gnu
yes |  createPackage.py dev-lang/python-3.8
```

Result:

```
Package name: (parameter) dev-lang/python-3.8
Create package with dependencies: (y/N)
dependencies=True
EDE version:              ede-3.8.0-next3

Package name: (full)      dev-lang/python-3.8.9
Continue: (Y/n)
working...
Toolchain name:           powerpc64-e5500-linux-gnu
 Depend packages:
app-arch/bzip2:0/1
app-arch/xz-utils:0/0
...
Package name: ede-3.8.0-next3--powerpc64-e5500-linux-gnu--dev-lang--python-3.8.9.tar.gz
Result: (in current directory)
ede-3.8.0-next3--powerpc64-e5500-linux-gnu--dev-lang--python-3.8.9.tar.gz
```

2.1.9 createMd5sumGeneric.py

The script `createMd5sumGeneric.py` creates md5sum from choosed directory.

```
createMd5sumGeneric.py -h
-d checking dir
-o output file
```

Example:

```
createMd5sumGeneric.py -d /opt/cov-analysis-linux64-2020.03/
-o /etc/ede/modules/x--cov-analysis-linux64-2020.03/cov-analysis-linux64-2020.03.md5
[ start ] create output file
[ start  ] generate md5 sum
...
[  ok  ] create output file
[  ok   ] generate md5 sum
```

Output file: */etc/ede/modules/x–cov-analysis-linux64-2020.03/cov-analysis-linux64-2020.03.md5*

```
565252bac4abcbddb44b80d43f6f4ded */opt/cov-analysis-linux64-2020.03/template-da/js/
node_modules/htmljs-parser/test/autotest/tag-name-expression-shorthand-id/expected.html
82d26f0c42e802bb81216dfe6e6609fa */opt/cov-analysis-linux64-2020.03/template-da/js/
node_modules/htmljs-parser/test/autotest/tag-name-expression-shorthand-id/input.htmljs
11788ee6ef9ef91810501108d55071d7 */opt/cov-analysis-linux64-2020.03/template-da/js/
node_modules/htmljs-parser/test/autotest/script-empty-concise/expected.html
ea3f19304ef51f99313e0959f8ebc398 */opt/cov-analysis-linux64-2020.03/template-da/js/
node_modules/htmljs-parser/test/autotest/script-empty-concise/input.htmljs
f4722f59560c25138fba56ce1f92cd75 */opt/cov-analysis-linux64-2020.03/template-da/js/
node_modules/htmljs-parser/test/autotest/shorthand-div-id-class/input.htmljs
92a5cdb1359e08ec54086f6dcad5e303 */opt/cov-analysis-linux64-2020.03/template-da/js/
node_modules/htmljs-parser/test/autotest/shorthand-div-id-class/expected.html
...
```

2.1.10 xkmake

The command *xkmake* is prepared for kernel compilation for embedded products.

```
ex.:
    xknew vm64
    xkmake vm64 mrproper
    xkmake vm64 clean
    xkmake vm64 oldconfig
    xkmake vm64 all -j$j
    xkmake vm64 modules -j$j
    xkmake vm64 modules_install
    xkmake vm64 install
```

xknew

Command *xknew* is for *xkmake*

2.2 Compilers

List available compilers:

```
gcc-config -l
```

List all compilers in EDE

```
 [1] aarch64-linux-gnu-7.3.0 *
 [2] armv7a-hardfloat-linux-gnueabi-7.3.0 *
 [3] i686-vm-linux-gnu-7.3.0 *
 [4] powerpc64-e5500-linux-gnu-7.3.0 *
 [5] powerpc-e500v2-linux-gnu-7.3.0 *
 [6] powerpc-e5500-linux-gnu-7.3.0 *
 [7] x86_64-pc-linux-gnu-6.4.0 *
 [8] x86_64-vm-linux-gnu-7.3.0 *
```

2.2.1 Native

The native compiler, which is available in *EDE tools*:

```
x86_64-pc-linux-gnu-*
```

Example:

```
x86_64-pc-linux-gnu-6.4.0
```

2.2.2 Cross

The cross compilers, which are available in EDE:

```
aarch64-linux-gnu-*
armv7a-hardfloat-linux-gnueabi-*
i686-vm-linux-gnu-*
powerpc64-e5500-linux-gnu-*
powerpc-e5500-linux-gnu-*
powerpc-e500v2-linux-gnu-*
x86_64-pc-linux-gnu-*
x86_64-vm-linux-gnu-*
```

[9]

[9] The *powerpc-e5500-linux-gnu-** is prepared only for u-boot build.

2.3 *EDE* external modules

External modules requirements

- In each *external module*, the `*.md5/sha*` must be included.
- Each user downloading external modules from *ARTI*[10] with *ADVA* domain credentials.
- *External modules* are not a part of the EDE.
- The archive with *external module* must be stored in *ARTI* [11] and should be consist of:

  ```
  <module name dir>/
    <module content>
    <module name>.md5
  ```

  The archiv will be extracted in */opt/**module name dir**/* directory in the *EDE*.
- Configuration module in *EDE* must be added by *EDE Team*

  ```
  /etc/ede/modules/x--<module name>/
   opt_e (arch extension, ex.: tar.bz2)
   opt_n (module name, ex.: jfrog)
   opt_p (path to ARTI, ex.:
    https://gdn-artifactory.rd.advaoptical.com/artifactory/tools/jfrogcli/jfrog-current.tar.bz2)
   opt_v (version, ex.: current)
   bashrc (JFROG_HOME=/opt/jfrog
           export PATH=$PATH:$JFROG_HOME)
  ```

  Verification:

  ```
  download_external_modules_from_arti.py jfrog
  ex.:
  download_external_modules_from_arti.py xilinx-2019.2 jfrog node-current
  ```

2.3.1 *EDE* external modules - solution for scripts

Also it's possible use *ede_rsync* with an external modules without interaction with user. For this case set environment variables are required: *artiUser* and *artiPass*.
Example:

```
# remove installed external modules first
rm /etc/ede/modules/x--node-current/wanted
rm /etc/ede/modules/x--jfrog/wanted
rm /etc/ede/modules/x--cov-analysis-linux64-current/wanted

ede_rsync --yes

# set environment variables
export artiUser=""
export artiPass=""

# install required external modules
touch /etc/ede/modules/x--jfrog/wanted
touch /etc/ede/modules/x--node-current/wanted
touch /etc/ede/modules/x--cov-analysis-linux64-current/wanted

ede_rsync --yes
```

[10] Internal storage for build tools

[11] It can be uploaded to Artifactory by Build Team only. *(SPE)*

Chapter 3

EDE embedded products

3.1 vm64

3.1.1 Evaluating operating system in virtualised x86 environment

EDE itself is not a runtime environment for ***AOS***, as it lacks some vital components, notably the kernel. It uses the host operating system's kernel, which is not standarised and therefore *AOS* behaviour would not be reproducible from one host to another.
EDE is a *cross-compilation* environment and should be used as such. To evaluate *AOS* components locally, use the *vm64*. It is a virtualised entity, having x86 (32-bit) architecture, running inside the host environment (qemu is used for virtualisation) and it is equipped with AOS-compatible kernel. Moreover, it has the same system software (especially libraries) in the same versions as other embedded target devices. From this point of view, *vm64* is just one of the target devices for *AOS*.

3.1.2 Required EDE modules

The *vm64* or *vm32* products requires choose three *EDE modules*: *arch–x86_64-vm-linux-gnu* (embedded), *prod–vm64* and for *i686* 32-bit Intel arch: *arch–i686-vm-linux-gnu.* [1]

Choosing modules by:

- script *ede_modules_config* or...
- manually touch empty files

  in */etc/ede/modules/...* and run *ede_rsync*.

  Example:

```
touch /etc/ede/modules/arch--x86_64-vm-linux-gnu/wanted
touch /etc/ede/modules/prod--vm64/wanted
touch /etc/ede/modules/arch--x86_64-vm-linux-gnu/wanted
# for arch i686
/etc/ede/modules/arch--i686-vm-linux-gnu/wanted
ede_rsync
```

3.1.3 Creation of virtual machine image

The main difference between *vm64* and physical devices is that instead of installation on physical device, a created virtual machine image is used directly by *qemu/kvm*. Such image has a form of directory, containing:

- kernel image as a file
- images of all volumes in some form (ext4 or 9p - see below)
- startup script, which is a wrapper around qemu *(when is used script vmrun only)*

[1] Currently *vm32* is not availabe.

- helper scripts for virtualised networking

  As said, volumes can be stored as either *ext4* or *9p* form. In *ext4* form, the volume is a file on the host, which is seen by the *vm64* as a block device containing *ext4* filesystem. In *9p* form, the volume is a subdirectory, which is shared with guest by *9p* protocol and seen as a network share. By default, volumes main and modules are *ext4*, all others are *9p*, but this can be altered at the *vm64* creation.

3.1.4 vm64 examples

After downloaded required modules, in *EDE* is available */targ/prod-devel/vm64* with content:

- *README.txt* with *how to's*
- product configuration scripts: *conf-vm64, conf-vm64-rootfs*
- kernel configuration files *(ex.: kernel_configs/v4.18-aufs/kernel_configuration-4.18)* [2]
- kernel mount point configuration: (for compilation) *kernel_sources/*
- *copy/scripts/deploy_hooks.py* example
- kernel build script: *build*

3.1.5 Create vm64 on disk

Invoke adva_deploy as follows:

```
    adva_deploy -i -d vm64 -f /path/to/configuration/file
```

The *configuration.file* for *vm64* is more complex than for other embedded architectures, because there are more degrees of freedom to handle; *vm64* is far less limited in terms of peripherals available than physical hardware. For example, number of networking interfaces is variable, as well as sizes of volumes. The example configuration file is as follows: (*(first simply example)*

```
specimen:
    ifs:
        eth0:
            ip: 192.168.32.2/24
            gw: 192.168.32.1
    dns:
    - 172.27.1.100
    - 172.27.1.15
    hostname: vm64-2
install:
    path: /home/jdoe/vm64/vm64-2
    overwrite: 1
```

Also it is possible to set parameters in command line, ex.: `{install.path: /home/jdoe/vm64/vm64-2}`.
In the section ***specimen*** there is specimen-specific data.
ifs section defines the networking interfaces. In the above example, *eth0* will be created.
dns and ***hostname*** entries work exactly as it looks.
In the ***install*** section, there are two parameters: ***path***, which specified the path where *vm64* image will be created, and *overwrite* which defines the behaviour if the directory is already present. In that case, if *overwrite* is *1* then it is overwritten, if *overwrite* is *0* then the installation fails with appropriate message.
volumes section defines type and size of volumes. By default, all volumes are in *ext4* form. If we want, for example, volumes *adva* and *rwd* to be *9p*, we add the *volumes* section as follows:

[2] The Linux kernel must be compiled with aufs module into kernel.

```
specimen:
    ifs:
        eth0:
            ip: 192.168.32.2/24
            gw: 192.168.32.1
    dns:
    - 172.27.1.100
    - 172.27.1.15
    hostname: vm64-2
install:
    path: /home/jdoe/vm64/vm64-2
    overwrite: 1
volumes:
    adva:
        type: 9p
    rwd:
        type: 9p
```

3.1.6 9p description

9p is a filesystem protocol used by *Plan 9* [3] operating system, but ported to some other operating systems as well, including *Linux. Linux* kernel has client functionality, whereas *qemu* has built-in server functionality, which does not rely on host operating system and even on networking at all. That is why it was chosen over *nfs, cifs* or other sharing filesystem types.

When *9p* is in use, *qemu* shares a host directory with guest operating system, so it is plainly accessible from both host and guest.

The sharing simplifies file exchange between host and guest. Instead of using scp or other network file transfer utilities, a file can be simply copied to the shared directory and is immediately visible by guest. The communication other way round is even simplier; as soon as the file is created in the guest *(on a volume which happens to be type 9p)*, it is accessible from host. This is especially valuable for logs. When logs are written by guest, they can be almost immediately read by host, using `tail -f` or similar mechanism. There is no need to copy them.

Unlike **ext4, 9p** has no rigid size. The host storage space occupied by a 9p share is roughly equals to the size of files in it - it is empty, it uses almost no host storage. Conversely, the emulated block device as a file occupies fixed size, even if it is empty. What makes it worse its size is its limit - after it becomes full, no further data can be written in it. With *9p*, the data can be written as long as there is free space on the host (or, more precisely, on host's partition in which the 9p share resides). This flexibility greatly saves space and is valuable especially for directories like */var.*

Another great advantage of *9p* shares over emulated block devices is that *9p* shares are continuously accessible when *vm64* starts and stops. With *ext4*, it's complicated. If *vm64* is not running, its *ext4* volume is accessible from host if mounted as loop block device. If *vm64* is to be started, it must be unmounted, otherwise it will lose integrity. Once *vm64* is started, it could be exported as *nfs* by *vm64* guest, but it would depend on networking between host and guest. And, anyhow, access to the *nfs* share would be broken if *vm64* stops. Then, it can be again mounted as loop device. With *9p*, the volume is always accessible from host, independently from guest power-cycling and restarting.

9p shares have also some disadvantages, compared to emulated block devices. One of them is worse performance. Another one is reduced capability set - some file operations are not supported by *9p.* If an application requires some unsupported operation on some file, it may be necessary to place the file in question on *ext4* volume.

Volumes *main* and *modules* are by default *ext4* because of performance reasons, because they contain rather static data, whose size is typically fixed and because they generally do not neded to be accessible from host. Other volumes are by default *9p*, to ease the development. It's simpler to put binaries to */opt/adva/**/bin* if it is shared, it is also simpler to read logs from */var/opt/adva/**/log* if it is shared.

[3] Plan 9 from Bell Labs is a distributed operating system, originally developed by the Computing Sciences Research Center at Bell Labs between the mid-1980s and 2002.

3.1.7 single disk mode for vm64

By default, *adva_deploy* for *vm64* creates as many virtual disks as many volumes it does have - i.e. every volume occupies a separate, unpartitioned disk. These disks are visible by guest as */dev/vda, /dev/vdb, /dev/vdc* and so on *(not /dev/sda, /dev/sdb, /dev/sdc and so on, because the disks are interfaced as VirtIO disks and not IDE).* This behaviour can be changed diametrally by using `-s onedisk` option:

```
adva_deploy -i -d vm64 -s onedisk -f configuration.file
```

This causes creation of a single, partitioned virtual disk *(with GPT)*, where every volume occupies a single partition. This disk is interfaced as IDE *(not as VirtIO)* and the partitions are seen by guest as */dev/sda1, /dev/sda2, /dev/sda3* and so on.

3.1.8 EFI boot

By default, *vm64* uses *qemu* bootloading feature. However, it is also possible to install the bootloader on the guest side, using *EFI* mechanism. This is activated by option `-s efi`:

```
adva_deploy -i -d vm64 -s efi -f configuration.file
```

`-s efi` implies `-s onedisk` and additionally installs *EFI* bootloader in the disk image. That way, the image is usable for hypervisors, which – contrary to qemu – do not have the bootloader functionality.

3.1.9 usage of the created virtual machine image

As said, the installation process creates a directory containing images of kernel, all volumes and some scripts. One of them, named vmrun, is used to start a virtual machine:

```
cd /path/to/virtual/machine/as/specified/in/configuration/file
sudo ./vmrun
```

Just running it starts the virtual machine and connects its *(emulated)* serial port to the console. This way we can observe the initialising kernel, then system services and then we can log in. Unless something is awfully misconfigured on the host side, we can also log in to the virtual machine via ssh, like to a physical device. We can start more than one instance of vitual machine, but we need a separate image directory for each of them. In other words, we must do separate installation, i.e. separate invocation of adva_deploy for each. Those virtual machines need to have unique *IP* addresses, otherwise there will be *IP* address conflict between them if they will be run simultaneously.
It is possible to run a virtual machine in the background, use `-s` option for this:

```
cd /path/to/virtual/machine/as/specified/in/configuration/file
sudo ./vmrun -s
```

This will use screen program for running the virtual machine in the background. You can at any time connect to the machine with:

```
cd /path/to/virtual/machine/as/specified/in/configuration/file
sudo ./vmrun -r
```

To detach, use `Ctrl+A, D` sequence, as in normal screen session. You can reconnect at any time.
To stop a virtual machine, log into it and issue halt. Be careful not to issue halt command in the host!

3.2 vm32

vm32 was a virtual machine which behaved mostly like vm64, except that it was 32-bit and is based on *i686-vm-linux-gnu* toolchain.

The product is available in repository *aos-ne-os* in place: *aos-ne-os/prod-devel-layer/targ/prod-devel/vm32*.
Required steps to generate disk image

- reho usage example:

```
export path2repos=<path to git repos dir>
mkdir -p $HOME/working/vm32-prod
```

- required repositories

  - aos-ne-os (checkouted on master)
  - f3-kernel-ne-pronid-vm *(currently kernel is used from vm64 product, but by this way can be mounted other kernels)*

- set alias:

```
alias rrVm32="sudo ./rehome/reho -r   \\
-e $path2repos/aos-ne-os/prod-devel-layer/targ/prod-devel/vm32/:/targ/prod-devel/vm32 \\
-e $HOME/working/vm32-prod/:/targ/prod/vm32/ \\
-e $path2repos/f3-kernel-ne-pronid-vm/: \\
                             /targ/prod-devel/vm32/kernel_sources/f3-kernel-ne-pronid-vm"
```

Required steps in EDE:

- 0) Required EDE modules in EDE: *ede_modules_config*
  choose: `arch--x86_64-vm-linux-gnu, arch--i686-vm-linux-gnu, prod--vm64, prod-devel--vm64`

- 1) syncing vm32 from product-devel to product

```
    cd /targ/prod-devel/vm32
    prod_copy_sync vm32
```

- 2) build kernel

```
    cd /targ/prod-devel/vm32
    ./build
```

  INFO:
  The current kernel source is set by symlink:

```
    ls -l kernel_sources/
    default_source -> linux-3.14.y/
```

  For this kernel, configuration file is set by symlink:

```
    ls -l kernel_configs/linux-3.14.y/
    default_config -> virtualbox_config
```

- 3) generate disk image

  use (or modify) product configuration file: *conf-vm32 adva_deploy:*

```
adva_deploy -i -d vm32  -s onedisk -s syslinux -s vga -f /targ/prod-devel/vm32/conf-vm32

# generate rootfs only
adva_deploy -i -d vm32 -f /targ/prod-devel/vm32/conf-vm32-rootfs
```

  [4] Before use in virtual box, please set chmod 777 on file: install-vm32/disk.img.vdi.

3.3 vm64/Fred

The product based on *vm64* and it is configured in *aos-product-fred* repository.

[4] Before use adva_deploy script, the kernel must be built.

3.3.1 Required modules/submodules in EDE

Fred product requires choose three *EDE modules*: *arch–x86_64-vm-linux-gnu (embedded), prod–vm64, arch–x86_64-nfv-linux-gnu (native).* Choosing modules by:

- script *ede_modules_config* or...
- manually touch empty files

  in */etc/ede/modules/...* and run *ede_rsync.*

  Example:

  ```
  touch /etc/ede/modules/arch--x86_64-vm-linux-gnu/wanted
  touch /etc/ede/modules/prod--vm64/wanted
  touch /etc/ede/modules/arch--x86_64-vm-linux-gnu/wanted
  touch /etc/ede/modules/arch--x86_64-nfv-linux-gnu/wanted
  ede_rsync
  ```

3.3.2 Fred repositories

Fred is configured in repositories:

- aos-product-fred
- aos-ne-tools

Product development is doing on *master* branch.

3.3.3 Minimal required steps to generate Fred image

- clone/checkout required git repositories

  Product Fred is configured in two repositories: *aos-product-fred, aos-ne-tools.*

- reho command

  reho mounts directories:

  - configuration product repository to: */targ/prod/Fred*
  - configuration product repository to: */targ/prod-devel/Fred*
  - configuration product repository to: */targ/prod/Fred/Doc*
  - working directory to: */mnt/working*

  configure exports for reho command

  ```
  export path2reposFred=<path to aos-product-fred-repositories>/ \\
                                          aos-product-fred/prod/Fred
  export path2reposFredDoc=<path to aos-product-fred-repositories>/ \\
                                          aos-product-fred/Doc
  export path2reposFredDevel=<path to aos-product-fred-repositories>/ \\
                                          aos-product-fred/prod-devel/Fred
  ```

Add packages for applications: VMM, Container Manager.(qemu, libvirt, lxc)

```
cd /targ/prod/Fred/
scripts/applyPackagesFromNFV.py Fred
```

call *reho* command:

```
sudo ./rehome/reho -r -e $path2reposFred/:/targ/prod/Fred \\
-e $path2reposFredDoc/:/targ/prod/Fred/Doc \\
-e $path2reposFredDevel/:/targ/prod-devel/Fred \\
-e <path to working directory>/working/:/mnt/working
```

3.3.4 Generate disk image

In this case, the configuration file should be prepared earlier and placed in */mnt/working* directory.

```
cd /mnt/working
adva_deploy -i -d Fred -s vagrant -f conf-fred
```

Examples: [5]

```
adva_deploy -i -d Fred -s vagrant -f conf-fred
adva_deploy -i -d Fred  -s onedisk -s syslinux -s vga -f /targ/prod/Fred/conf-fred
# VMM
adva_deploy -i -d Fred  -s onedisk -s syslinux -s vga -f /targ/prod/Fred/conf-fred-2-vmm
# generate rootfs only
adva_deploy -i -d Fred  -s onedisk -s syslinux -s vga -f /targ/prod/Fred/conf-fred-2-rootfs
```

3.3.5 Generate rootfs (for LXC)

Required modules: *arch–x86_64-vm-linux-gnu, prod–vm64, prod-devel–vm64*
Generating rootfs can be usefull to create package or use it in the container *(LXC)*.

rootfs vm64

```
# generate rootfs only
cd /targ/prod-devel/vm64/
adva_deploy -i -d vm64 -f conf-vm64-rootfs
or (with full path)
adva_deploy -i -d vm64 -f /targ/prod-devel/vm64/conf-vm64-rootfs
```

Results:

```
result path: /output/products/install-vm64/rootfs
```

3.3.6 Working on generated rootfs vm64

After generate the rootfs for vm64, it's possible to working as on the VM [6], example:

```
# /rehome/reho -r -d /output/products/install-vm64/rootfs
gdn-n-przemekg / #
```

From this moment the rootfs is */output/products/install-vm64/rootfs*. The option *-e* for mount points definition, also can be used, example:

```
# /rehome/reho -r -d /output/products/install-vm64/rootfs -e /mnt/gitRepos/:/mnt/gitRepos/
gdn-n-przemekg / # ls -l /mnt/gitRepos/
drwxrwxr-x 18 1000 1000 4096 Aug 27 08:31 aos-ne-os
```

rootfs Fred

All repositories for *Fred* product must be used. 3.3

```
# generate rootfs only
adva_deploy -i -d Fred  -s onedisk -s syslinux -s vga -f /targ/prod/Fred/conf-fred-3-rootfs
```

Required section in configuration file:

[5] *-s vagrant* should be used only in *ede-2.2.x*.
[6] virtual machine

```
proxy:
    path: /output/products/proxy_for_vm64
    stay: 1
    only: 1
    result: install-fred/rootfs
    overwrite: 1
```

Section `only: 1` is required.

3.4 The product set up

Reference design script is available in */targ/prod-devel/vm64/copy/scripts/deploy_hooks.py*.
Product configuration files examples:

- for rootfs: */targ/prod-devel/vm64/conf-vm64-rootfs*
- for disk img: */targ/prod-devel/vm64/conf-vm64*

The directory */targ/prod-devel/vm64/* consist of all required directories and files which should be created to prepare a new product.

- kernel build script: *build*
- the *copy* directory, consist of all scripts which will be copied to */targ/prod/* directory:

  arch.txt, consist of architecture information

  mode.txt, consist of configuration mode information. (default is eos)

  other tools, example *vmdir*

More examples is stored in the *aos-ne-os* repository in directory: *prod-devel-layer/targ/prod-devel*.

3.5 Apply product configuration

- before use *adva_deploy*
  It's described in section *"product configuration file"*. 2.1.7
- after use *adva_deploy.*
  Applying a product configuration is possible by *"system-overrides"* mechanism directly on products. [7]
  How it works: The software package should consist of configurtaion files, which should be extracted on working machine to: */opt/adva/system-overrides/etc/.*

  From this moment the new configuration files are visible in */etc/* directory.

3.6 Manage users and groups on products

[8]

Adding a new users and groups is possible by two ways:

- when disk image is generated, by *adva_deploy* command,
- in installed software package on box.

[7] Not applicable F7 product.
[8] This feature is available since ede-3.7.0.

3.6.1 Configuring users and groups while generating disk image

The directory with *fsmods (currently: aos_users)* for users and groups definitions *(mount point in EDE)* is required in product repository, but in *EDE*, it should be mounted in:

```
/targ/prod-devel/vm64/copy/fsmods/aos_users
```

Directory content: *(example, fsmods name: aos_users)*
(full path: /targ/prod-devel/vm64/copy/fsmods/aos_users/opt/adva/system-overrides/etc/synth_ns)

```
ls -l opt/adva/system-overrides/etc/synth_ns
total 16
drwxr-xr-x 2 root root 4096 Apr 22 06:24 group
drwxr-xr-x 2 root root 4096 Apr 22 06:26 gshadow
drwxr-xr-x 2 root root 4096 Apr 22 06:22 passwd
drwxr-xr-x 2 root root 4096 Apr 22 06:26 shadow
```

User id must be in range: 1000-1500
passwd

```
cat  /opt/adva/system-overrides/etc/synth_ns/passwd/01passwd
aos:x:1000:1000:aos
```

group

```
cat  /opt/adva/system-overrides/etc/synth_ns/group/01group
aos:x:1000:
```

shadow

```
cat  /opt/adva/system-overrides/etc/synth_ns/shadow/01shadow
aos:!:::::::
```

gshadow

```
cat  /opt/adva/system-overrides/etc/synth_ns/gshadow/01gshadow
aos:::aos
```

Product configuration: *(required section in product configuration file)*

```
fsmods_local:
- aos_users
```

3.6.2 Configuring users and groups in software package

In this case, the software packages should consist of users definitions: (directories with file contents)

```
ls -l /opt/adva/system-overrides/etc/synth_ns
total 16
drwxr-xr-x 2 root root 4096 Apr 22 06:24 group
drwxr-xr-x 2 root root 4096 Apr 22 06:26 gshadow
drwxr-xr-x 2 root root 4096 Apr 22 06:22 passwd
drwxr-xr-x 2 root root 4096 Apr 22 06:26 shadow
```

[9]

passwd

```
cat  /opt/adva/system-overrides/etc/synth_ns/passwd/02passwd
aos:x:1000:1000:aos
```

group

```
cat  /opt/adva/system-overrides/etc/synth_ns/group/02group
aos:x:1000:
```

shadow

```
cat  /opt/adva/system-overrides/etc/synth_ns/shadow/02shadow
aos:!:::::::
```

gshadow

```
cat  /opt/adva/system-overrides/etc/synth_ns/gshadow/02gshadow
aos:::aos
```

Result *after reboot* on the product:

```
cat /etc/passwd
aos:x:1000:1000:aos

cat /etc/group
aos:x:1000:

cat /etc/shadow
aos:!:::::::

cat /etc/gshadow
aos:::aos
```

Configuring users and groups by standard commands directly on *Operating System*

Configure users and groups also is possible in runtime directly on working operating system by standard commands: *useradd, groupadd, usermod.*

useradd

```
    useradd
Usage: useradd [options] LOGIN
       useradd -D
       useradd -D [options]

Options:
      --badnames                do not check for bad names
  -b, --base-dir BASE_DIR       base directory for the home directory of the
                                new account
      --btrfs-subvolume-home    use BTRFS subvolume for home directory
  -c, --comment COMMENT         GECOS field of the new account
```

[9] The files numbering *(02passwd, 02groups, 02shadow, 02gshadow) - 02** should be different than used earlier while generating disk image process. *(if would be the same, it will be replaced, not merged).*

```
  -d, --home-dir HOME_DIR       home directory of the new account
  -D, --defaults                print or change default useradd configuration
  -e, --expiredate EXPIRE_DATE  expiration date of the new account
  -f, --inactive INACTIVE       password inactivity period of the new account
  -g, --gid GROUP               name or ID of the primary group of the new
                                account
  -G, --groups GROUPS           list of supplementary groups of the new
                                account
  -h, --help                    display this help message and exit
  -k, --skel SKEL_DIR           use this alternative skeleton directory
  -K, --key KEY=VALUE           override /etc/login.defs defaults
  -l, --no-log-init             do not add the user to the lastlog and
                                faillog databases
  -m, --create-home             create the user's home directory
  -M, --no-create-home          do not create the user's home directory
  -N, --no-user-group           do not create a group with the same name as
                                the user
  -o, --non-unique              allow to create users with duplicate
                                (non-unique) UID
  -p, --password PASSWORD       encrypted password of the new account
  -r, --system                  create a system account
  -R, --root CHROOT_DIR         directory to chroot into
  -P, --prefix PREFIX_DIR       prefix directory where are located the /etc/* files
  -s, --shell SHELL             login shell of the new account
  -u, --uid UID                 user ID of the new account
  -U, --user-group              create a group with the same name as the user
```

groupadd

```
    groupadd
Usage: groupadd [options] GROUP

Options:
  -f, --force                   exit successfully if the group already exists,
                                and cancel -g if the GID is already used
  -g, --gid GID                 use GID for the new group
  -h, --help                    display this help message and exit
  -K, --key KEY=VALUE           override /etc/login.defs defaults
  -o, --non-unique              allow to create groups with duplicate
                                (non-unique) GID
  -p, --password PASSWORD       use this encrypted password for the new group
  -r, --system                  create a system account
  -R, --root CHROOT_DIR         directory to chroot into
  -P, --prefix PREFIX_DIR       directory prefix
```

usermod

```
    usermod
Usage: usermod [options] LOGIN

Options:
  -b, --badnames                allow bad names
  -c, --comment COMMENT         new value of the GECOS field
  -d, --home HOME_DIR           new home directory for the user account
  -e, --expiredate EXPIRE_DATE  set account expiration date to EXPIRE_DATE
  -f, --inactive INACTIVE       set password inactive after expiration
                                to INACTIVE
  -g, --gid GROUP               force use GROUP as new primary group
  -G, --groups GROUPS           new list of supplementary GROUPS
  -a, --append                  append the user to the supplemental GROUPS
                                mentioned by the -G option without removing
                                the user from other groups
  -h, --help                    display this help message and exit
  -l, --login NEW_LOGIN         new value of the login name
  -L, --lock                    lock the user account
  -m, --move-home               move contents of the home directory to the
                                new location (use only with -d)
  -o, --non-unique              allow using duplicate (non-unique) UID
  -p, --password PASSWORD       use encrypted password for the new password
  -R, --root CHROOT_DIR         directory to chroot into
  -P, --prefix PREFIX_DIR       prefix directory where are located the /etc/* files
  -s, --shell SHELL             new login shell for the user account
  -u, --uid UID                 new UID for the user account
  -U, --unlock                  unlock the user account
  -v, --add-subuids FIRST-LAST  add range of subordinate uids
  -V, --del-subuids FIRST-LAST  remove range of subordinate uids
  -w, --add-subgids FIRST-LAST  add range of subordinate gids
  -W, --del-subgids FIRST-LAST  remove range of subordinate gids
```

3.7 Enable quota on the partition

[10]

EDE delivers frontend to simply quota configuration with pre-defined partitions for that. *(rwda, rwd)*
Commands added by EDE:

- quota_init
- quota_set

quota_init is for initialize quota in ***OS***[11]
quota_set activates quotas on mount points (disks)
In software, *quotas* can be activated by script: *(example: set_quota.sh)*

```
------------------------>8-----------------------
  #!/bin/bash -xe

  quota_init
  quota_set <user_name> 1000 1000 50 50 rwda
------------------------8<-----------------------
```

Command *quota_set* set quota on */mnt/active* and it uses directly command setquota with all accepted parameters.

```
INFO:
setquota --help
setquota: Usage:
  setquota [-u|-g|-P] [-F quotaformat] <user|group|project>
    <block-softlimit> <block-hardlimit> <inode-softlimit> <inode-hardlimit> -a|<filesystem>...
  setquota [-u|-g|-P] [-F quotaformat]
                <-p protouser|protogroup|protoproject> <user|group|project> -a|<filesystem>...
  setquota [-u|-g|-P] [-F quotaformat] -b [-c] -a|<filesystem>...
  setquota [-u|-g|-P] [-F quotaformat] -t <blockgrace> <inodegrace> -a|<filesystem>...
```

[10] This feature is available since ede-3.6.9.

[11] Operating System

```
  setquota [-u|-g|-P] [-F quotaformat]
                <user|group|project> -T <blockgrace> <inodegrace> -a|<filesystem>...

-u, --user                 set limits for user
-g, --group                set limits for group
-P, --project              set limits for project
-a, --all                  set limits for all filesystems
    --always-resolve       always try to resolve name, even if is
                           composed only of digits
-F, --format=formatname    operate on specific quota format
-p, --prototype=protoname  copy limits from user/group/project
-b, --batch                read limits from standard input
-c, --continue-batch       continue in input processing in case of an error
-t, --edit-period          edit grace period
-T, --edit-times           edit grace times for user/group/project
-h, --help                 display this help text and exit
-V, --version              display version information and exit
```

Chapter 4

EDE software development

4.1 Working with source code

Working with source code requires do some configuration sets. The path to repositories should be shared in *EDE* preferably on the same path like on host. It will be useful for *gdb* [1] usage.

4.1.1 mount source code

Parameter to *reho* command: *(-e)*
-e [path on host]:[path in EDE]
Example:

```
-e /mnt/gitRepos/:/mnt/gitRepos
```

By this way it is possible to use more mount points in *EDE*.
Example:

```
sudo rehome/reho -r -e /mnt/gitRepos/:/mnt/gitRepos
    -e /mnt/gitRepos/f3-kernel-ne-pronid-vm/:/targ/prod-devel/vm64/kernel_sources/v4.18-aufs
```

The each *mount point* in *EDE* must be created before use it.

4.1.2 cross compilers usage

Show list available *cross-compilers*: *(in EDE)*

```
# gcc-config -l
 [1] aarch64-linux-gnu-7.3.0 *
 [2] armv7a-hardfloat-linux-gnueabi-7.3.0 *
 [3] i686-vm-linux-gnu-7.3.0 *
 [4] powerpc-e500v2-linux-gnu-7.3.0 *
 [5] powerpc64-e5500-linux-gnu-7.3.0 *
 [6] x86_64-pc-linux-gnu-7.3.0 *
 [7] x86_64-vm-linux-gnu-7.3.0 *
```

- *x86_64-pc-linux-gnu*, it's a native compiler available in *EDE* tools.
- *x86_64-vm-linux-gnu*, it's a cross compiler for target architecture *x86_64-vm-linux-gnu*. Currently it's used in *vm64/Fred* product.
- *powerpc64-e5500-linux-gnu*, it's a cross compiler for target architecture *powerpc64-e5500-linux-gnu*. Currently it's used in *ECM* products. [2]
- *powerpc-e500v2-linux-gnu*, it's a cross compiler for target architecture *powerpc-e500v2-linux-gnu*. Currently it's used in *F7-NCU* products.

[1] GDB: The GNU Project Debugger
[2] In all *ECM's*.

- *aarch64-linux-gnu*, it's a cross compiler for target architecture *aarch64-linux-gnu*. Currently it's used in *F4* product.
- *armv7a-hardfloat-linux-gnueabi*, it's a cross compiler for target architecture *armv7a-hardfloat-linux-gnueabi*.
- *i686-vm-linux-gnu*, it's a cross compiler for target architecture *i686-vm-linux-gnu*. Currently it's used in tests 32 bit software in *F7* project.

4.2 Working with chroot/eval *(reho with -d option)*

Required modules: *arch–x86_64-vm-linux-gnu*
Using *reho* command with *-d* parameter sets *rootfs* in a current environement. Additional it is possible use *-e* parameter to mount external shares. Example:

```
-d  -e /mnt/gitRepos/:/mnt/gitRepos
```

Full one command:

```
sudo rehome/reho -r -e /mnt/gitRepos/:/mnt/gitRepos/
     -d ‘pwd‘/targ/arch/x86_64-vm-linux-gnu/modes/eos
```

Result:

```
# mount
...
/dev/vda1 on /mnt/gitRepos type ext4 (rw,relatime,errors=remount-ro)
```

It's also possible do it in two steps:

- `sudo rehome/reho -r -e /mnt/gitRepos/:/mnt/gitRepos/`
- `rehome/reho -r -d /targ/arch/x86_64-vm-linux-gnu/modes/eos/ -e /mnt/gitRepos/:/mnt/gitRepos/`

Conclusion: by this way we can work directly on generated embedded target, but fully compatible environments requires generated *rootfs* with specified packages per product, it is described in next section.
Working environement requires two terminals:

- first with *EDE Tools*: `sudo rehome/reho -r -e /mnt/gitRepos/:/mnt/gitRepos/` for software compilators
- second with embedded target *x86_64* for ex.: *unit tests*

4.2.1 run unit tests in chroot/eval environements *(directly on target vm64 in EDE)*

Required modules: *arch–x86_64-vm-linux-gnu, prod–vm64, prod-devel–vm64*

4.2.2 run unit tests in generated *rootfs*

Required modules: *arch–x86_64-vm-linux-gnu, prod–vm64, prod-devel–vm64*
Required steps:

- generate *rootfs* 3.3.5
- set *-d* parameter (path to generated rootfs) to *reho* command:

  ```
  Example:
  -d /output/products/install-vm64/rootfs
  ```

4.2.3 gdb best practice

gdb test configuration

- generate rootfs 3.3.5
- in first terminal:

  ```
  cd <path to EDE>
  sudo rehome/reho -r  -e /mnt/gitRepos/:/mnt/gitRepos
  /rehome/reho -r -d /output/products/install-vm64/rootfs -e /mnt/gitRepos/:/mnt/gitRepos
  gdbserver --multi  localhost:10000
  ```

- in second terminal:

  ```
  cd <path to EDE>
  sudo rehome/reho -r  -e /mnt/gitRepos/:/mnt/gitRepos
  gdb
  (gdb) target remote localhost:10000
  Remote debugging using localhost:10000
  ```

- in first terminal get:

  ```
  Remote debugging from host 127.0.0.1
  ```

- example on ECM

  ```
  ecm ~ # gdb
  GNU gdb (Gentoo 8.2.1 p1) 8.2.1
  ...
  This GDB was configured as "powerpc64-e5500-linux-gnu".
  Type "show configuration" for configuration details.
  ...
  (gdb) show configuration
  This GDB was configured as follows:
     configure --host=powerpc64-e5500-linux-gnu --target=powerpc64-e5500-linux-gnu
               --with-auto-load-dir=$debugdir:$datadir/auto-load
               --with-auto-load-safe-path=$debugdir:$datadir/auto-load
               --without-expat
               --with-gdb-datadir=/usr/share/gdb (relocatable)
               --with-jit-reader-dir=/usr/lib/gdb (relocatable)
               --without-libunwind-ia64
               --without-lzma
               --without-babeltrace
               --without-intel-pt
               --disable-libmcheck
               --without-mpfr
               --without-guile
               --with-separate-debug-dir=/usr/lib/debug (relocatable)

  ("Relocatable" means the directory can be moved with the GDB installation
  tree, and GDB will still find it.)

  ecm cd...

  ecm /usr/lib/debug/sbin # gdb
  ...
  ```

```
This GDB was configured as "powerpc64-e5500-linux-gnu".
...
(gdb) symbol-file sln.debug
Reading symbols from sln.debug...(no debugging symbols found)...done.
```

[3]

4.3 Introduction to Linux kernel development

EDE set up environment

- *ede_modules_config*

  choose *arch–x86_64-vm-linux-gnu, prod–vm64*

- build kernel

  ```
  cd /targ/prod-devel/vm64
  ./build
  ```

- compile external module [4]

  clean:

  ```
  xkmake vm64 clean \\
  M=/mnt/gitRepos/aos-ne-os/Documentation/ede-2.x.x/examples/external_kernel_module/hello/
  ```

- module:

  ```
  xkmake vm64 modules \\
  M=/mnt/gitRepos/aos-ne-os/Documentation/ede-2.x.x/examples/external_kernel_module/hello/

    CC [M]  /mnt/gitRepos/... /hello.o
    Building modules, stage 2.
    MODPOST 1 modules
    CC      /mnt/gitRepos/... /hello.mod.o
    LD [M]  /mnt/gitRepos/... /hello.ko
  ```

- results

  ```
  ls /mnt/gitRepos/<external module path>/external_kernel_module/hello/
    hello.c
    hello.ko
    hello.mod.c
    hello.mod.o
    hello.o
    Makefile
    modules.order
    Module.symvers
  ```

- test module

[3] The GDB requires set architecture, ex.: ARCH=powerpc64-e5500-linux-gnu

[4] Simple external kernel module is available in repository:
aos-ne-os/Documentation/ede-3.x.x/examples/external_kernel_module/hello/.

```
insmod /lib/modules/hello.ko

lsmod
hello                   12496  0

dmesg
...
66.663101] Hello, world
...
```

Chapter 5

EDE for Workgroups ;)

5.1 EDE multi user usage

- each user should have account on the server [1]
- *EDE* should be installed in generic location

  ```
  ex.: /opt/ede-3.x.x/ (this directory must be created earlier)
  (rsync -ai rsync://10.143.218.2/ede/base/3.6.0/.  /opt/ede-3.x.x/.)
  ```

- choose required modules *(ede_modules_config)*
- mount points configure

  In EDE the mount points for home users should be created,

  ```
  ex: sudo mkdir -p /opt/ede-3.x.x/home/jminer
  ```

  The chown on home directories should be set also,

  ```
  ex.: sudo chown jminer.jminer /opt/ede-3.x.x/home/jminer
  ```

- first logon

  First logon as user to EDE is required,

  ```
  ex.:sudo rehome/reho -r -u jminer
  (user will be added  to passwd and groups)
  ```

- mount required shares in EDE

  The mount point is required and should be created earlier,

  ```
  ex.:
  sudo mkdir -p  /opt/ede-3.x.x/home/jminer/gitRepos/
  sudo chown jminer.jminer /opt/ede-3.x.x/home/jminer/gitRepos/
  ```

  Now it is possible to mount the shared dirs from host in EDE (form hosts, the same location – it is important for code debugging...)

  ```
  sudo  /opt/ede-3.x.x/rehome -r -u jminer -e /home/jminer/gitRepos/:/home/jminer/gitRepos/
  ```

Editing the sources is possible on host or in *EDE. GIT* also can be used in *EDE*, but should be configured. *(git user, ssh keys)*

[1] Server for product development dedicated for the team. (not EDE server)

5.2 EDE with ssh access on the server

For this purpose, the EDE should be installed in generic location on the server and *ssh daemon* should be started while *OS* was starting. In *EDE* is available script, which could be used on the server for this functionality.

- required users in the shared EDE, must be added to host *(server)*
- port must be different than 22
- minimum required parameter is three. *EDE_PATH, SSHD_PORT, "User or users"*
- Users must be defined as parameter in quotation marks.

For test, it can be started by hand of course.

```
<ede path>/opt/adva/bin/ede_for_workgroups/ede_sshd_start.sh <ede path> <port> "<user1> <usr2> ..."

Ex.: (run as root)
/opt/ede-3.x.x/opt/adva/bin/ede_for_workgroups/ede_sshd_start.sh /opt/ede-3.x.x 7022 "john bld wrk"

Result:
root@gdn-vc-ede-test-01:/opt/ede-3.x.x#
/opt/adva/bin/ede_for_workgroups/ede_sshd_start.sh /opt/ede-3.x.x 7022 pguza
Params: /opt/ede-3.x.x 7022 pguza
3
adding element: pguza
pguza
[ start ] create users
/opt/ede-3.x.x
7022
pguza
[ start ] copy_etc_shadow
/opt/ede-3.x.x
pguza
pguza:$y$j9T$eDrYjtuJk.PbqQf2gFGFQ/$66kxfkh2fB7BZHlvIrh76dpAkV33MJRviM/ndRMf7H0:18891:0:99999:7:::
INFO: The user pguza is available in /etc/shadow!
[  ok   ] copy_etc_shadow
pguza
[  ok   ] create users
/opt/ede-3.x.x
7022
[ start ] EDE as server
    [ start ] set up configuration
7022
    [  ok   ] set up configuration
[ start ] set sshd port
[  ok   ] set sshd port
[ start ] generate red group
[  ok   ] generate red group
[ start ] start sshd
[  ok   ] start sshd
[  ok   ] EDE as server
PID sshd information:
  1  579  579  579 ?  -1 Ss  0  0:00 sshd: /usr/sbin/sshd [listener] 0 of 10-100 startups
```

From this moment, the *EDE* is available by ssh client.

```
pguza@gdn-vc-przemyslawg-01:$ ssh -p 7022  192.168.1.165
```

To *.profile* stored in home directories is added line:

```
source /etc/profile
```

The */home/users* directories are available in the *EDE*.