ACAP Developer Guide - Version 3.1.0 - 3.4.2

SDK

ACAP Developer Guide - Version 3.1.0 - 3.4.2

1 Introduction

AXIS Camera Application Platform (ACAP) is Axis own open application platform. It provides a development platform for software-based solutions and systems built around Axis devices. It's available on various types of Axis products, and not only on our cameras.

ACAP makes it possible to develop applications for a wide range of applications:

- Security applications that improve surveillance systems and facilitate investigation.
- Business intelligence applications that improve business efficiency.
- Product feature plug-ins that add value beyond the Axis product's core functionality.

1.1 Key features

AXIS Camera Application Platform SDK (ACAP SDK) provides:

- Full access to most common product features, including video, audio, the event system, I/O ports, PTZ control, and more.
- Enables hardware acceleration on specific functions such as image analysis computations, overlay graphics and more.
- Support for multiple architectures to build applications for all ACAP-enabled Axis products.
- Example applications for all APIs to kickstart your development.

1.2 This is the ACAP SDK

The ACAP SDK consists of:

ACAP SDK – This Docker container image contains all tools for building and packaging an ACAP 3 application as well as API components (header and library files) needed for accessing different parts of the camera firmware. The toolchain and API images are also available as separate images, described below. This guide assumes that you are working with the combined image.

ACAP Toolchain – This Docker container image contains the tools, compilers and scripts required to build applications.

ACAP API – This Docker container image contains header files, libraries and other API components that you need when building an ACAP application.

1.3 This is why we use Docker

From version 3.1, the SDK format relies on the *Docker* toolchain, something we see many developers use to manage their software. An important advantage of using Docker is that the SDK can be decoupled from the underlying host operating system. This enables us to support development across all Docker capable platforms, including not only Linux, but also Windows 10 and MacOS X. Docker also offers a rich set of tools to build and ship software.

If you're new to Docker and its many features, check out their *Getting Started guide*.

Follow the instructions in *Get started on page 5* to start developing your ACAP using Docker and our containerized SDK.

1.4 Compatibility

Compatibility means that if an ACAP can be installed and run on a specific device, then the ACAP is compatible with the device. Compatibility depends on both hardware and software (firmware).

1.4.1 Hardware compatibility

ACAP is supported on a large portion of all Axis devices. For an ACAP to be hardware compatible with a specific device, the ACAP must be compiled using the SDK corresponding to the chip architecture in the device.

ACAP version 3 SDK is available for two different architectures, armv7hf (32 bit) and aarch64 (64 bit).

1.4.2 Software compatibility

An ACAP is software compatible with an AXIS OS firmware if the APIs and other ACAP runtime features are available in the specific firmware release. The available APIs in a firmware depends on both the firmware version and the device itself since some APIs are only relevant for certain devices. For example, the Video capture API is only available on devices with an image sensor.

1.4.3 Find the right SDK for hardware compatibility

Find the right SDK for the hardware architecture of the specific device or devices that you want to develop for.

Chip	Architecture
ARTPEC-6	armv7hf
ARTPEC-7	armv7hf
S2E	armv7hf
S2L	armv7hf
S3L	armv7hf
S5	aarch64
S5L	aarch64

See detailed instruction in *Find out which SDK to use on page 5* .

1.4.4 Find the right SDK for software compatibility

Choose the appropriate SDK version based on what firmware version you want supporting your ACAP.

[SDK versions and related compatible AXIS OS firmware versions]

SDK version	Compatible with firmware version
SDK 3.0	9.70 and later
SDK 3.1	9.80 (LTS) and later
SDK 3.2	10.2 and later
SDK 3.3	10.5 and later
SDK 3.4	10.6 and later

1.4.4.1 Forward compatibility

The ACAP is forward compatible for the firmware related to a specific SDK version. This means that the ACAP is compatible for the listed firmware version and future firmware versions until the next firmware LTS (Long Term Support). After an LTS, there may be breaking changes that breaks compatibility, for example when a deprecated API is removed. Breaking changes are always announced in advance.

An ACAP built with an SDK that is based on an older firmware version should always work on a newer firmware version within the same LTS window.

1.4.4.2 Feature growth between LTS releases

To get new features, always use the latest ACAP SDK release. A new feature could be, for example, a new version of an API.

New SDK versions between LTS releases always add functionality in a way that an ACAP, built using a previous version, will still compile with the new version of the SDK.

Read more about AXIS OS release tracks and related information *here*.

1.4.4.3 Supporting older firmware

If you want an ACAP to be compatible with older firmware, you need to choose an SDK for an older firmware.

2 Get started

To get started using ACAP:

1. Set up your developer environment. See *ACAP development requirements*.
2. *Find out which SDK to use*.
3. *Set up and verify the SDK on page 10*.

2.1 ACAP development requirements

To develop ACAPs, you need the following:

- A computer with Linux, Windows, or MacOS.
- Docker. *Get Docker*.
  - For Windows and MacOS, use the automatically upgradable **Docker desktop stable**.
  - For Linux, use **Docker community edition** version 18.09 or later.
- GIT scm. *GIT downloads*.
- We also recommend using Microsoft **Visual Studio Code**. But you are free to choose your favorite editor.

2.2 Find out which SDK to use

Different products have different architectures depending on the chip used in the product. Consequently different SDKs are required to ensure that the ACAP is compatible with the product. Currently, there are two different SDKs - one for products with **armv7hf** architecture, and one for products with **aarch64** architecture.

Choosing which SDK to use depends on the starting point of your development:

I already have a specific product that I want to develop an ACAP for – Which SDK to use depends on the architecture of the specific product. To find out which architecture your product has, see *Check device properties on page 7* .

I want to develop an ACAP for a specific product that I don't have access to yet – Find out which chip a specific product has in *Product interface guide* (requires login to Developer Community).

I don't have a specific product, or I'm not sure which architecture to use. – If you're not sure, use **armv7hf**, which supports most products.

Note

ACAP supports the chips listed in the *Product interface guide*.

2.3 Set up the device

To prepare a device for development:

- *Find the device on the network* to configure IP address and user credentials.
- *Check device compatibility* to make sure that you use a device that supports ACAP.
- *Check device properties* and update to the latest firmware if needed, see *How to upgrade*.
- *Check device properties* and identify the device architecture to be able to choose the correct toolchain.

2.3.1 Find the device on the network

To find Axis devices on the network and assign them IP addresses in Windows®, use AXIS IP Utility or AXIS Device Manager. Both applications are free and can be downloaded from *axis.com/support*.

For more information about how to find and assign IP addresses, go to *How to assign an IP address and access your device*.

2.3.1.1 Access the device

1. Open a browser and enter the IP address or host name of the Axis device.

   If you do not know the IP address, use AXIS IP Utility or AXIS Device Manager to find the device on the network.

2. Enter the username and password. If you access the device for the first time, you must set the root password. See *Set a new password for the root account on page 6*.

3. The live view page opens in your browser.

2.3.1.2 Verify that no one has tampered with the firmware

To make sure that the device has its original Axis firmware, or to take full control of the device after a security attack:

1. Reset to factory default settings. See the product's user manual for information on how to reset to factory default settings.

   After the reset, secure boot guarantees the state of the device.

2. Configure and install the device.

2.3.1.3 Secure passwords

Important

Axis devices send the initially set password in clear text over the network. To protect your device after the first login, set up a secure and encrypted HTTPS connection and then change the password.

The device password is the primary protection for your data and services. Axis devices do not impose a password policy as they may be used in various types of installations.

To protect your data we strongly recommend that you:

- Use a password with at least 8 characters, preferably created by a password generator.
- Don't expose the password.
- Change the password at a recurring interval, at least once a year.

2.3.1.4 Set a new password for the root account

Important

The default administrator username is **root**. If the password for root is lost, reset the device to factory default settings.

Support tip: Password security confirmation check

1. Type a password. Follow the instructions about secure passwords. See *Secure passwords on page 6*.
2. Retype the password to confirm the spelling.
3. Click **Create login**. The password has now been configured.

2.3.2 Check device compatibility

To check if device supports ACAP, use VAPIX CGI `http://192.168.0.90/axis-cgi/param.cgi` and check if the device supports EmbeddedDevelopment with a VAPIX GET/POST request and action list.

To extract the value of the parameter, use the CGI as device URL in a browser window:

```
http://192.168.0.90/axis-cgi/param.cgi?action=list
&group=Properties.EmbeddedDevelopment.EmbeddedDevelopment
```

The following response shows a device that supports ACAP:

```
root.Properties.EmbeddedDevelopment.EmbeddedDevelopment=yes
```

2.3.3 Check device properties

To get basic device information, such as firmware version and architecture, use VAPIX POST request and method **getAllProperties**:

```
http://192.168.0.90/axis-cgi/basicdeviceinfo.cgi
```

To extract the messages, use the CGI from a terminal, using the credentials set in the network configuration:

```
curl --anyauth "*" -u [username]:[password] 192.168.0.90/axis-cgi/basicdeviceinfo.cgi
--data "{\"apiVersion\":\"1.0\",\"context\":\"Client defined request ID\",\"method
\":\"getAllProperties\"}"
```

The following response contains architecture `"Architecture": "armv7hf"`, and firmware version `"Version": "9.50.1"`:

```
{
     "apiVersion": "1.0",
     "context": "Client defined request ID",
     "data": {
     "propertyList": {
          "Architecture": "armv7hf",
          "Brand": "AXIS",
          "BuildDate": "Nov 07 2019 11:16",
          "HardwareID": "764",
          "ProdFullName": "AXIS P1448-LE Network Camera",
          "ProdNbr": "P1448-LE",
          "ProdShortName": "AXIS P1448-LE",
          "ProdType": "Network Camera",
          "SerialNumber": "ACCC8ED0C9EC",
          "Soc": "Axis Artpec-6",
```

```
            "SocSerialNumber":  "00000000-00000000-442C2402-87C00153",
            "Version":  "9.50.1",
            "WebURL":  "http://www.axis.com"
            }
      }
}
```

2.3.4 How to upgrade

Axis offers product firmware management according to the active track or the long-term support (LTS) tracks. Regardless of the track chosen, it's recommended to upgrade the firmware regularly in order to get the latest security updates. The firmware can be upgraded using AXIS Device Manager, AXIS Camera Station, AXIS Companion, HTTP or FTP.

Note

- If using AXIS A1001 in cluster mode, make sure to upgrade all controllers. Either all at a time using AXIS Device Manager or straight after each other using the web interface or FTP. The entire cluster should always be on the same firmware.
- The following Axis devices require a firmware upgrade via the latest available LTS 2018 (8.40) prior to upgrading to AXIS OS 10.9 or later: AXIS D2050-VE, AXIS FA54, AXIS M3057-PLVE, AXIS M3058-PLVE, AXIS P1367/-E, AXIS P1368-E, AXIS P1445-LE, AXIS P1445-LE-3, AXIS P1447-LE, AXIS P1448-LE, AXIS P3227-LV/-LVE, AXIS P3228-LV/-LVE, AXIS P3807-PVE, AXIS Q1645/-LE, AXIS Q1647/-LE, AXIS Q1659, AXIS Q3515-LV/-LVE, AXIS Q3517-LV/-LVE/-SLVE, D101-A XF P3807, ExCam XF P1367, ExCam XF Q1645 and F101-A XF P1367.

2.3.4.1 AXIS Device Manager or AXIS Camera Station

1. Go to the **Device Manager Tab** in Axis Device Manager or **Configuration Tab > Devices – Management** in AXIS Camera Station.
2. Select all devices that should be upgraded.
3. Right click and select **Upgrade Firmware**, which will open a dialog with a list of device models and the current firmware version installed in each device.
4. In the **Upgrade Firmware** dialog there are two options:
   - **Check for Updates** which requires internet access.
   - **Browse** to locate firmware file e.g. on hard drive or memory stick.
5. Check for updates:
   - Select **Check for Updates** to download a list of all firmware available for all device models.
6. Browse:
   - Select **browse** to locate firmware files and import them. It is possible to select and import several files at the same time.
7. Each device model will show a dropdown containing all available firmware for a model. Firmware will be sorted by "Long Term Support" (LTS) and "Active" software tracks in the dropdown.
8. Select the firmware to install for each device model.
9. Click **OK** to start upgrading the devices.

Note

By default, firmware upgrade is done for all the selected devices at the same time. The upgrade order can be changed in **Configuration > Connected services > Firmware upgrade settings**. Once a firmware update has been started, the devices will be unavailable until the installation and restart of the devices has completed successfully.

For more information, see the help in AXIS Device Manager/AXIS Camera Station.

2.3.4.2 HTTP

Note

The procedure to update firmware differs slightly depending on the version of the installed web interface (before and after 7.10.1). For AXIS Q1659, the new web interface was introduced in firmware version 6.55.1.

Upgrade instructions when using the old web interface

1. Download the upgrade file to a directory that is accessible from your local computer.
2. Open the product's start page (e.g. `http://192.168.0.90`) in a web browser.
3. Click the **Setup** link and log in as "root" (or any other user with administrator privileges). You must be logged in as an administrator to upgrade the unit.
4. Click **System Options** in the menu to the left.
5. Click **Maintenance**.
6. Click the **Browse** button in the **Upgrade Server** section.
7. Select the upgrade file you downloaded (and maybe decompressed) from our site. This file is typically named after the product and firmware version.
8. Click the **Open** button.
9. Click the **Upgrade** button in the **Upgrade Server** section.
10. Wait for the flash load to complete, which may take 1-10 minutes. The upgrade procedure occurs in four steps:
    - Step 1: Shut down. Running applications are shut down and active connections are terminated.
    - Step 2: Uploading firmware. The old firmware will be erased and the new firmware will be saved. During this step, the power LED will blink green/amber. After a while the progress of the upgrade will be displayed in the web browser.
    - Step 3: Reboot. The system restarts automatically.
    - Step 4: Reconfiguration. The new firmware settings are configured to match the previous settings. The color of the status LED will be amber during this step.
11. After the upgrade has completed, the unit will automatically initiate the system, during which the status LED blinks amber. When initiation is complete and the system is ready for use, the status LED will be green.

Upgrade instructions when using the new web interface

1. Download the upgrade file to a directory that is accessible from your local computer.
2. Open the product's start page (e.g. `http://192.168.0.90`) in a web browser.
3. Log in as "root" (or any other user with administrator privileges).
4. Click **Settings** to the right.
5. Click **System-tab**.
6. Click **Maintenance**.
7. Select the upgrade file you downloaded (and maybe decompressed) from our site. This file is typically named after the product and firmware version.
8. Click the **Open** button.

9. Click the **Upgrade** button in the **Upgrade Server** section.
10. Wait for the flash load to complete, which may take 1-10 minutes. The upgrade procedure occurs in four steps:
    - Step 1: Shut down. Running applications are shut down and active connections are terminated.
    - Step 2: Uploading firmware. The old firmware will be erased and the new firmware will be saved. During this step, the power LED will blink green/amber. After a while the progress of the upgrade will be displayed in the web browser.
    - Step 3: Reboot. The system restarts automatically.
    - Step 4: Reconfiguration. The new firmware settings are configured to match the previous settings. The color of the status LED will be amber during this step.
11. After the upgrade has completed, the unit will automatically initiate the system, during which the status LED blinks amber. When initiation is complete and the system is ready for use, the status LED will be green.

2.3.4.3 FTP

Note

- Starting with firmware version 7.30.1 and onwards, the FTP server is disabled by default. In order to use the instructions below it first needs to be enabled via the web interface: **Settings > System > Plain config > Network > NetworkFTP**
- This section is not applicable for AXIS Companion Line cameras.

1. You must be at the command prompt and in the directory that contains the upgrade file.

   Example: `C:\Axis\Product\Firmware`
2. From the command prompt, open an FTP connection to the unit you wish to upgrade.
   (Do not use a Windows based FTP program to do this, use command line FTP programs only.)
3. Log in as "root". You must be logged in as the root user to upgrade the unit.
4. Change to binary transfer mode by typing "bin" and press enter.
5. Type "hash" and press enter. This will allow you to see how the upgrade progresses.
6. Type the command "put XXX.bin flash" where XXX.bin is the name of the upgrade file you downloaded (and maybe decompressed) from our site. This file is typically named after the product and firmware version.
7. Wait for the flash load to complete, which may take 1-10 minutes. The upgrade procedure occurs in four steps:
    - Step 1: Shut down. Running applications are shut down and active connections are terminated.
    - Step 2: Uploading firmware. The old firmware will be erased and the new firmware will be saved. During this step, the power LED will blink green/amber. After a while, the progress of the upgrade will be displayed in the command prompt.
    - Step 3: Reboot. The FTP session terminates and the system starts automatically.
    - Step 4: Reconfiguration. The new firmware settings are configured to match the previous settings. The color of the status LED will be amber during this step.
8. After the upgrade has completed, the unit will automatically initiate the system, during which the status LED blinks amber. When initiation is complete and the system is ready for use, the color of the status LED will be green.

2.4 Set up and verify the SDK

The ACAP SDK image, available on *Docker Hub*, is based on Ubuntu and contains the environment needed for building an AXIS Camera Application Platform (ACAP) application. It includes all tools for building and packaging an ACAP 3 application as well as API components (header and library files) needed for accessing different parts of the camera firmware.

The SDK image can be used as a basis for custom built images to run your application, or as a developer environment inside the container.

To set up and verify the SDK, build, install and run a Hello World application example below.

1. *Create a Hello World application on page 11*
2. *Build the Hello World application on page 13*
3. *Install and run the Hello World application on page 13*

If you want to Build and install the Hello World application inside container, see *Build and install the Hello World application inside container on page 14*.

After completing the **Hello World** application example, you are set to start with more complex examples, see *Code examples and tutorials on page 26* for more information.

Check out the sections *Application project structure on page 16* and *Build, install, and run the application on page 21* for further details about building, running and installing applications.

2.4.1 Create a Hello World application

Create the following folder and file structure in a working directory:

```
hello-world
├── app
│   ├── hello_world.c
│   ├── LICENSE
│   ├── Makefile
│   └── manifest.json
├── Dockerfile
└── README.md
```

The files comprising the following:

hello_world.c

Hello World application which writes to system-log.

```
#include <syslog.h>
int main(int argc, char **argv)
{
/* Open the syslog to report messages for "hello_world" */
openlog("hello_world", LOG_PID | LOG_CONS, LOG_USER);
openlog("hello_world", LOG_PID | LOG_CONS, LOG_USER);
/* Choose between { LOG_INFO, LOG_CRIT, LOG_WARN, LOG_ERR }*/
syslog(LOG_INFO, "Hello World!");
syslog(LOG_INFO, "Hello World!");
/* Close application logging to syslog */
closelog();
closelog();
return 0;
}
```

LICENSE

Text file that lists all open source licensed source code distributed with the application.

Note

If **LICENSE** is empty the build fails.

Makefile

Makefile containing the build and link instructions for building the ACAP application.

Note

Make sure to preserve the tabs below. Recipes in a makefile must be preceded by a single standard tab character.

```
PROG1 = hello_world
OBJS1 = $(PROG1).c
OBJS1 = $(PROG1).c
PROGS = $(PROG1)
PROGS = $(PROG1)
CFLAGS-y = -W -Wformat=2 -Wpointer-arith -Wbad-function-cast -Wstrict-prototypes
-Wmissing-prototypes -Winline -Wdisabled-optimization -Wfloat-equal -Wall -Werror
CFLAGS-y = -W -Wformat=2 -Wpointer-arith -Wbad-function-cast -Wstrict-prototypes
-Wmissing-prototypes -Winline -Wdisabled-optimization -Wfloat-equal -Wall -Werror
all:  $(PROGS)
all:  $(PROGS)
$(PROG1):  $(OBJS1)
     $(CC) $^ $(CFLAGS) $(LIBS) -o $@
     $(STRIP) $@
$(STRIP) $@
clean:
     rm -f $(PROGS) *.o core *.eap
```

manifest.json

Defines the application and its configuration.

```
{
"schemaVersion":  "1.1",
"acapPackageConf":  {
"setup":  {
"appName":  "hello_world",
"vendor":  "Axis Communications",
"embeddedSdkVersion":  "3.0",
"user":  {
"username":  "sdk",
"group":  "sdk"
},
"runMode":  "never",
"version":  "1.0.0"
}
}
}
```

Dockerfile

Docker file with the specified Axis toolchain and API container to build the example specified.

```
ARG ARCH=armv7hf
ARG VERSION=3.3
ARG UBUNTU_VERSION=20.04
ARG REPO=axisecp
ARG SDK=acap-sdk
ARG SDK=acap-sdk
FROM ${REPO}/${SDK}:${VERSION}-${ARCH}-ubuntu${UBUNTU_VERSION}
FROM ${REPO}/${SDK}:${VERSION}-${ARCH}-ubuntu${UBUNTU_VERSION}
FROM ${REPO}/${SDK}:${VERSION}-${ARCH}-ubuntu${UBUNTU_VERSION}
# Building the ACAP application
COPY ./app /opt/app/
WORKDIR /opt/app
RUN . /opt/axis/acapsdk/environment-setup* && acap-build ./
```

2.4.2 Build the Hello World application

Standing in your working directory run the following commands:

```
docker build --tag <APP_IMAGE> .
```

<APP_IMAGE> is the name to tag the image with, e.g., hello_world:1.0

Default architecture is **armv7hf**. To build for **aarch64** it's possible to update the `ARCH` variable in the Dockerfile or to set it in the docker build command via build argument:

```
docker build --build-arg ARCH=aarch64 --tag <APP_IMAGE> .
```

Copy the result from the build to a local directory `build`.

```
docker cp $(docker create <APP_IMAGE>):/opt/app ./build
```

The working directory now contains a build folder with the following files:

```
hello-world
├── app
│   ├── hello_world.c
│   ├── LICENSE
│   ├── Makefile
│   └── manifest.json
├── build
│   ├── hello_world*
│   ├── hello_world_1_0_0_armv7hf.eap
│   ├── hello_world_1_0_0_LICENSE.txt
│   ├── hello_world.c
│   ├── LICENSE
│   ├── Makefile
│   ├── manifest.json
│   ├── package.conf
│   ├── package.conf.orig
│   └── param.conf
├── Dockerfile
└── README.md
```

build/hello_world* – Application executable binary file.

build/hello_world_1_0_0_armv7hf.eap – Application package .eap file

build/hello_world_1_0_0_LICENSE.txt – Copy of LICENSE file.

build/manifest.json – Defines the application and its configuration.

build/package.conf – Defines the application and its configuration.

build/package.conf.orig – Defines the application and its configuration, original file.

build/param.conf – File containing application parameters.

2.4.3 Install and run the Hello World application

To install your application on an Axis video product:

1. Browse to the following page (replace `<axis_device_ip>` with the IP number of your Axis video product).

   `http://<axis_device_ip>/#settings/apps`

2. Click Add, the **+** icon, and browse to the newly built `hello_world_1_0_0_armv7hf.eap`.
3. Click **Install**.

`hello_world` is now available as an application on the device.

To run the application:

1. Browse to the following page (replace `<axis_device_ip>` with the IP number of your Axis video product).

   `http://<axis_device_ip>/#settings/apps`

2. Click the application icon.
3. **Start** the application.

2.4.4 Expected output from the Hello World application

You can find the application log at:

`http://<axis_device_ip>/axis-cgi/admin/systemlog.cgi?appname=hello_world`

or by clicking the **App log** link in the device GUI.

You can also extract the logs using the following commands in the terminal:

Note

Make sure SSH is enabled on the device to run the following commands.

```
tail -f /var/log/info.log | grep hello_world
```

Output

```
----- Contents of SYSTEM_LOG for 'hello_world' -----
----- Contents of SYSTEM_LOG for 'hello_world' -----
14:13:07.412 [ INFO ] hello_world[6425]:  Hello World!
```

2.4.5 Build and install the Hello World application inside container

Standing in your working directory, run the following command to bind mount the application directory in to the acap-sdk container:

```
docker run --rm -v $PWD/app:/opt/app -it hello_world:1.0
```

Or if you want to use the original image:

```
docker run --rm -v $PWD/app:/opt/app -it axisecp/acap-sdk:3.3-armv7hf-ubuntu20.04
```

Now inside the container, to build the application, run:

```
acap-build ./
```

The app directory now contains the following files:

```
├── Dockerfile
└── app
├── LICENSE
├── Makefile
├── hello_world*
├── hello_world.c
├── hello_world.o
├── hello_world_1_0_0_LICENSE.txt
├── hello_world_1_0_0_armv7hf.eap
├── manifest.json
├── package.conf
├── package.conf.orig
└── param.conf
```

To install the application to a camera use command:

```
eap-install.sh --help
```

3 Develop applications

3.1 Application project structure

An application project contains several files and directories for an application. The mandatory files are:

File name	Description
[application name].(c/cpp)	Source file/files in C/C++.
LICENSE	Text file that lists all open source licensed source code distributed with the application.
Makefile	Defines how the source code is compiled, includes dependencies and sets compilation error levels.
manifest.json	• Defines the application and its configuration. • For ACAP SDK version 3.3 and later. • Used at installation of the package. See *Create a manifest file from scratch on page 17* and *Create a manifest file from existing package.conf on page 18* for more information.
package.conf	• Defines the application and its configuration. • For ACAP SDK version 3.2 and earlier. • Used at EAP file creation and at installation of the package. See *Package configuration file on page 20* for more information.

Note

- Use **manifest.json** for ACAP SDK version 3.3 and later.
- Use **package.conf** for ACAP SDK version 3.2 and earlier
- An eap package based on **manifest.json** is similar to one based on **package.conf**. The features previously configured using **package.conf** and special configuration files are now included in **manifest.json**.

Other optional files and directories to include:

File Name	Type	Description
lib	Directory	Shared libraries custom to the application.
Postinstall Script	Shell Script	Executed at the installation of the application.
Preupgrade Script	Shell Script	Executed before upgrading the application, available from firmware version 9.30.

3.1.1 Manifest file

manifest.json defines the application and its configuration.

An eap package based on **manifest.json** is similar to one based on a **package.conf**. The features previously configured using the **package.conf** and special configuration files are now included in **manifest.json**.

Use **manifest.json** for ACAP SDK version 3.3 and later.

Note

For ACAPs built using **manifest.json**, license key handling is supported from firmware version 10.6 and later.

For more information see:

- *Create a manifest file from scratch on page 17*
- *Create a manifest file from existing package.conf on page 18*
- *Manifest file content on page 18*
- *Manifest file schema on page 19*
- *Discontinued support when using manifest file on page 19*

3.1.1.1 Create a manifest file from scratch

Create a **manifest.json** file based on the manifest.json schema, see *Manifest file schema on page 19*.

To create the manifest file for a simple Hello Glib ACAP application:

1. Create a minimal manifest.json file with basic metadata:

- Friendly name
- Identifier and binary
- Vendor name
- Version

Example 1:

```
"friendlyName":  "Hello Glib",
"appName":  "hello_glib",
"vendor":  "Axis Communications",
"version":  "2.0.0"
```

2. Define how you want the applications to be executed:

- Running mode - the application keeps running at a reboot.
- User and group for execution and file ownership, typically the sdk:sdk.
- If needed, you could also define special start options for the execution of the binary.

Example 2:

```
"runMode":  "never",
"user":  {
     "group":  "sdk",
     "username":  "sdk"
}
```

3. Add the required embedded development version on the target device.

Example 3:

```
"embeddedSdkVersion":  "2.0"
```

4. Add any supported cgi endpoints.

Example 4:

```
"configuration":  {
     "httpConfig":  [
          {
               "access":  "viewer",
```

```
                "name":  "example.cgi",
                "type":  "transferCgi"
            }
        ]
}
```

The finished manifest.json

```
{
    "schemaVersion":  "1.0",
    "acapPackageConf":  {
        "setup":  {
            "friendlyName":  "Hello Glib",
            "appName":  "hello_glib",
            "vendor":  "Axis Communications",
            "version":  "2.0.0",
            "embeddedSdkVersion":  "2.0",
            "runMode":  "never",
            "user":  {
                "group":  "sdk",
                "username":  "sdk"
            },
        },
        "configuration":  {
            "httpConfig":  [
                {
                    "access":  "viewer",
                    "name":  "example.cgi",
                    "type":  "transferCgi"
                }
            ]
        }
    }
}
```

3.1.1.2 Create a manifest file from existing package.conf

If there is a **package.conf** file for the ACAP application project, you can use it to generate an **initial manifest.json** file.

The easiest way to do this is interactively inside the container. To generate the initial **manifest.json** file:

1. Locate **packageconf2manifest.py** tool inside the container, under **/opt/axis/acapsdk/axis-acap-manifest-tools**.
2. Run **packageconf2manifest.py** inside your ACAP project (that has an existing **package.conf** file).

This generates a manifest file based on that configuration. The manifest file should include everything that is needed to build the EAP file.

For help on using **packageconf2manifest**, run `packageconf2manifest.py -h`.

3.1.1.3 Manifest file content

The table below shows the package configuration with manifest file, in relationship with the package.conf file.

Setting	With manifest file	In package conf and files
Application identifier and main binary	acapPackageConf.setup.appName	APPNAME
The user friendly name of the application	acapPackageConf.setup.friendlyName	PACKAGENAMEMENUNAME
Name of the application vendor	acapPackageConf.setup.vendor	VENDOR
	acapPackageConf.setup.vendorUrl	VENDORHOMEPAGELINK

Setting	With manifest file	In package conf and files
Version for the ACAP application	acapPackageConf.setup.version	APPMAJORVERSION
		APPMICROVERSION
		APPMINORVERSION
User and group for execution and file ownership	acapPackageConf.setup.user.username	APPUSR
	acapPackageConf.setup.user.group	APPGRP
Id of the application in Axis ACAP portal	acapPackageConf.setup.appId	APPID
The execution behavior of the application	acapPackageConf.setup.runMode	STARTMODE
Special options for binary execution	acapPackageConf.setup.runOptions	APPOPTS
Open source licenses file	file: LICENSE	file: LICENSE
Copy protection solution	copyProtection.method	LICENSEPAGE
Application specific installation script	acapPackageConf.installation.postInstallScript	POSTINSTALLSCRIPT
Required embedded development version	acapPackageConf.setup.embeddedSdkVersion[1]	REQEMBDEVVERSION
Application specific setting page	acapPackageConf.configuration.settingPage	SETTINGSPAGEFILE
Supported cgi endpoints	acapPackageConf.configuration.httpConfig[].name acapPackageConf.configuration.httpConfig[].type acapPackageConf.configuration.httpConfig[].access	HTTPCGIPATHS and file
Product integrated application parameters	acapPackageConf.configuration.paramConfig[].default acapPackageConf.configuration.paramConfig[].name acapPackageConf.configuration.paramConfig[].type	file: param.conf
Application specific web client pages	folder: html/	folder: html/
Application specific dynamically linked libraries	folder: lib/	folder: lib/

1. `2.0` for manifest.json schema version 1.0 and earlier (for firmware version 10.5 and earlier). `3.0` for manifest.json schema version 1.1 and later. The minor version may need to be stepped up for certain APIs. See *API on page 29* for more information.

3.1.1.4 Manifest file schema

You can find the schema **application-manifest-schema-v1.1.json** in the container under **/opt/axis/acapsdk/axis-acap-manifest-tools/schema/schemas**. Use version 1.0 **application-manifest-schema-v1.0.json** for compatibility with older firmware (10.4 and earlier).

3.1.1.5 Discontinued support when using manifest file

The following is no longer supported for an ACAP application, when using manifest file:

Pre-upgrade script – Script run before an ACAP application is installed as an upgrade.

Forking main process – Forking and demonizing of the ACAP application during start.

Note

Instead of the application forking the main process, the application is started by systemd, which assumes failure if the main process dies, and writes any output to stdout/stderr to the system log.

3.1.2 Package configuration file

The **package.conf** file is included in the application package and contains variables describing the application. Shell script command `create-package.sh` reads the contents of **package.conf** when the application package is created.

3.1.2.1 Mandatory parameters

Below are descriptions of the mandatory parameters in the **package.conf** file.

Parameter	Description
APPGRP	The group that the application will run as. The recommended group is **sdk**.
APPID	The application copy protection identifier. Provided by Axis and required when using Axis copy protection solution.
APPMICROVERSION	A numerical value of the application's micro version.
APPMINORVERSION	A numerical value of the application's minor version.
APPMAJORVERSION	A numerical value of the application's major version.
APPNAME	The name of the application's executable binary file.
APPOPTS	Contains the application command line options (may be empty).
APPTYPE	The (generated) architecture for which the package is built.
APPUSR	The user that the application will run as. The recommended user is **sdk**.
LICENSENAME	Specifies **LICENSE** file information text.
LICENSEPAGE	Specifies if a copy protection solution is used by the application. Possible values are: • `axis` Protected by Axis copy protection license solution. • `custom` Protected by a custom copy protection license solution, not provided by Axis. • `none` Not protected by any license copy protection license solution.
OTHERFILES	A space-separated list (may be empty) of other files and/or directories to be included in the package. Files listed here are copied to the application directory during installation. OTHERFILES can be used if separate libraries or configuration files are used by the main program.
PACKAGENAME	A user-friendly package name, to be used by for instance device management software, when referring to the application.
REQEMBDEVVERSION[1]	Specifies the minimum required embedded development version that the device running the application must support. The version is dependent on the set APIs used in the application.
STARTMODE	Defines how the application is started. Possible values are: • `respawn` Once started, the application starts automatically when the system starts (at boot). In case the application crashes, it restarts automatically. • `once` Once started, the application starts automatically when the system starts (at boot). In case the application crashes, does not restart. • `never` Application does not start or restart automatically.
VENDOR	The name of the vendor that created the application, to be used by for instance device management software, when referring to the vendor.

1. `2.0` if you are using package.conf only (and not manifest.json). The minor version may need to be stepped up for certain APIs. See *API on page 29* for more information.

3.1.2.2 Optional parameters

Below are descriptions of the optional parameters in the **package.conf** file.

Parameter	Description
PREUPGRADESCRIPT	A script that runs on the device before upgrading the application, with the purpose of preparing the installed application for the upgrade.
POSTINSTALLSCRIPT	A script that runs on the device when the installation is completed.
SETTINGSPAGEFILE	Specifies the html page for custom settings, to be used by for instance device management software, allowing a user to browse the application settings page. The file must be in a directory called html in the application project.
VENDORHOMEPAGELINK	Specifies a link to the vendor's homepage, to be used by for instance device management software, when referring to the vendor.

3.1.3 License file

The **LICENSE** file is included in the application package. It shall contain all required open source license information for open source code distributed with the application package.

Note

If **LICENSE** is empty the build fails.

3.1.4 Local data

Application data such as configuration data and images should in runtime be saved to the localdata directory of the application project's root directory. The localdata directory is owned by APPUSR and the application has write access to it, once installed on a device. All content in the localdata directory remains after an application upgrade.

NOTICE

Avoid continuous writes to **localdata** as it resides on a flash drive with limited write count per block. Internal wear leveling minimizes the risk of failure, however, it is still strongly recommended to avoid continuous writes.

Note

The available free space is product dependent.

3.2 Build, install, and run the application

To build, install and run an ACAP application, use the image in the *axisecp/acap-sdk* repository. You can use this image in two ways:

- As a base image to build a custom application builder image, see *Build, install and run with custom application image on page 22*.
- Interactively inside the container, see *Build, install and run interactively inside container on page 23*.

To build an application defined by manifest.json, use the **acap-build** tool, see *Build tool on page 22*.

To build an application defined by package.conf, use the **create-package.sh** build script, see *Build script on page 21*.

3.2.1 Build script

For applications defined by **package.conf**, use the build script **create-package.sh**.

The script does the following:

- Checks that the file **package.conf** exists and does not contain any errors.
- Asks for missing or invalid parameters and creates **package.conf**.

- Executes make in the current directory to compile the application source code into an application binary file.
- Creates an **eap** (embedded application package) package including the application binary, html files (if any) and configuration files.
- Creates a copy of `LICENSE` file.

The created application package filename has the following format:

```
<PACKAGENAME>_<APPMAJORVERSION>_<APPMINORVERSION>_<APPMICROVERSION>_<SDK_ARCHITECTURE>.eap
```

The created copy of `LICENSE` filename has the following format:

```
<PACKAGENAME>_<APPMAJORVERSION>_<APPMINORVERSION>_<APPMICROVERSION>_LICENSE.txt
```

3.2.2 Build tool

For applications defined by **manifest.json**, use the build tool **acap-build**.

The acap-build tool does the following:

- Runs make, performing any required cross-compilation as defined in the available Makefile.
- Validates the manifest file against the manifest schema.
- Generates a package.conf file and related configuration files for backward compatibility.
- Creates an EAP file including:
  - application executable
  - LICENSE file
  - any available html and lib folder
  - other files listed in the acap-build request
  - generated backward compatibility files

For help on using the build tool, run `acap-build -h`.

Note

- If any additional files were previously listed in **OTHERFILES** in the **package.conf** file, these need to be listed as input to the acap-build command using the flag -a, for example `acap-build ./ -a file1 -a file2`.
- At some point an EAP will be required to be based on a manifest file instead of a **package.conf** file. For such an ACAP application package to be supported in older firmware, a **package.conf** file is generated and included in the EAP file. Although it's the manifest file that is the base setup file for the ACAP application when building an EAP package in the SDK.
- In the next step of introducing manifest file EAP files, systemd will start and stop the ACAP application. It then assumes execution failure if the main process dies, which means that the process must not fork off to a background process.

3.2.3 Build, install and run with custom application image

For instructions on how to set up your build, to install, and to run with custom application image, use the **Hello World** application example shown in *Set up and verify the SDK on page 10*.

Using the custom application image, all the building and packaging is done inside a Docker container. The application is then copied to a custom directory, meaning that the original application project directory is not changed.

The top structure for an ACAP application contains a Dockerfile and a directory called `app` where the application project files are placed.

To install, start, stop and remove the application, use the device's web interface.

3.2.4 Build, install and run interactively inside container

To be able to work interactively with your application, you can bind mount the application project directory into the container. In this way, you can build and package the application directly in this folder. You may also install, start, stop and remove the application on a device directly from inside the container.

The top structure for an ACAP application contains a directory called, for example `app` where the application project files are placed.

To run the acap-sdk container interactively and mount the application project, go to the directory that contains `app` and run:

```
docker run -v $PWD/app:/opt/app -it hello_world:1.0
```

where:

- `axisecp/acap-sdk` is the Docker hub repostitory
- `3.3-armv7hf-ubuntu20.04` is the tag that points out which SDK version and architecture to use
- `-v $PWD/app:/opt/app` mounts the host directory `$PWD/app` into the container directory `/opt/app`
- `--rm` removes the container after closing it
- `-i` is to run the container interactively
- `-t` which repository tag to use

You should end up in a container with a prompt similar to:

```
root@1e6b4f3d5a2c:/opt/app#
```

Now you're ready to build and install the application. See *Build the application on page 23*, and *Install the application on page 23*

Note

The bind mount means that any changes made inside the container on `/opt/app` will be made to the host directory `$PWD/app`.

3.2.4.1 Build the application

Using package.conf (ACAP SDK version 3.2 and earlier)

To build an application stand in the application directory inside the container and run **create-package.sh**.

Using manifest.json (ACAP SDK version 3.3 and later)

To build an application, stand in the application directory inside the container and run the **acap-build** tool.

3.2.4.2 Install the application

The SDK helps with installing a built application on the device from a terminal. You can also install application packages, using the device's web interface. But this method is less convenient during application development.

To install a built application on a device, run:

```
eap-install.sh
```

Run the command without any options to get help.

To install a built application on a device, run the following command (you must enter the IP address and the root password of the device the first time):

```
eap-install.sh <device-ip> <password> install
```

```
eap-install.sh 192.168.0.90 pass install
```

The command remembers the device-ip and password after the first successful execution. After this you can simply run:

```
eap-install.sh install
```

Note

You must run the command from the application project directory, see *Application project structure*.

3.2.4.3 Start, stop, and remove the application

Before you continue, make sure that you have done a first successful execution of shell script command `eap-install.sh`, see *Install the application* for more information.

To start, stop and remove an installed application, run:

```
eap-install.sh [start|stop|remove]
```

To start an installed application on the device, run:

```
eap-install.sh start
```

Now you can see the status of the application using the device's web interface.

To stop a running application, run:

```
eap-install.sh stop
```

To remove an installed application, run:

```
eap-install.sh remove
```

3.3 Reproducible builds

Reproducible builds is a set of practices for compiling software that ensures the resulting binary code can be reproduced.

See *An example of how to create a reproducible ACAP application* to learn more.

3.4 Develop using Visual Studio Code

Visual Studio Code provides access to containerized development tools, without the need for you to install them natively on your development computer.

To start developing ACAPs using Visual Studio Code:

1. Install the **Remote - Containers** extension available in the **Extensions Marketplace** in **Visual Studio Code**.
2. Create a subfolder called `.devcontainer` in the top directory of the source code project you are working on.
3. In `.devcontainer`, create a Dockerfile and `devcontainer.json` containing the code below.
4. Save the Dockerfile and `devcontainer.json`.
5. Press **Ctrl+Shift+P** and type in `Remote-Containers: Reopen in Container`.

Dockerfile

```
FROM axisecp/acap-sdk:3.2-armv7hf-ubuntu20.04
CMD /bin/bash
```

devcontainer.json

```
{
      "name":  "ACAP SDK ",
      "build":  {
            "dockerfile":  "Dockerfile"
```

```
        }
}
```

The application restarts and is now attached to a container with the SDK and your code. This way, you can interactively edit your source code just as if the tools had been installed natively, including using all the debugging and support features in Visual Studio Code.

You can install different versions of the SDK in separate containers. When you open your source code folder, Visual Studio Code identifies the SDK version defined in the Dockerfile.

The ACAP SDK-container includes all the SDK tools, Git and some other useful things. But you can add more tools to the Dockerfile and the `devcontainer.json` configuration. See *Microsofts tutorials on how to use Development Containers* for more information on what this way of working can offer.

3.5 Supported languages

The ACAP SDK is capable of building applications in various languages.

Shell script

The build process requires a Makefile, even if nothing is being compiled. An empty Makefile is necessary to build shell script programs.

C

Most of the examples are built using C and can be found on *GitHub*. The SDK uses gcc to compile C programs.

C++

For an example of a C++ application, go to *using-opencv*. The SDK uses g++ to compile C++ programs.

4 More resources

4.1 Code examples and tutorials

Several ACAP code examples and tutorials are open-sourced and available on *Github*.

Each example has a well-defined README file with the example structure on GitHub which will help you execute the examples on an Axis camera. The README file structure is comprised of:

- example description
- repository structure
- limitation
- how to build and run the code
- expected output

The tutorials have the same structure as the examples but are more extensive in content and scope.

We are continuously adding new examples and tutorials for both existing as well as for new functionality. Keep checking GitHub to stay updated with the latest changes.

4.2 Developer community

Axis Developer Community is an Axis hosted website for developers that want to work with Axis devices. Membership is required but it's open to anyone.

On the community pages you'll find

- a wide array of documentation and tools that are useful when developing ACAP applications.
- documentation on, for example, how to integrate with Axis devices and other SDKs.
- a reference guide for all Axis devices supporting ACAP with information about chipset and memory resources.
- a Developer loan tool for running tests and trying out Axis products online.

4.3 Product interface guide

The *Product interface guide* gives you information about architecture and chipset memory resources (RAM and flash) for all supported Axis devices. This information is helpful for finding out which SDK to use.

5 Services for partners

5.1 Package signing

The sign package service is available in the ACAP Service Portal, and requires Technology Integration Partner Program access.

By signing an ACAP package you make sure that the content of the package is not tampered with between the release and installation on the Axis device.

During the signing process, a signature is added at the end of the package. The signature is verified by the device when installing the ACAP.

Support for verifying signed ACAPs was introduced in firmware 9.20. The format as such is fully backward compatible. A signed ACAP can be installed on devices with a firmware earlier than 9.20, in which case it's not verified by the device.

5.2 ACAP Service Portal for administrators

Use the ACAP Service Portal for administrators to register new applications compatible with Axis products, and to manage all related information such as:

- Name and description for applications.
- Enable and setup the licensing service, such as trial or free licenses to users.
- Manage all aspects around licenses such as generating codes, and track and modify license keys.
- Configure compatibility between your application and Axis products.
- Sign ACAP packages to ensure authenticity, and to prevent tampering.

5.2.1 Access the ACAP Service Portal

Note

You must be an *Technology Integration Partner* to access the ACAP Service Portal.

1. Go to *axis.com* and sign into your personal axis account.
2. Go to **My Axis** > **Partner Pages**.
3. Go to **Software Development** > **ACAP** > **Axis ACAP Service Portal**.

Note

All your colleagues with Technology Integration Partner Program access have access to the ACAP Service Portal per default. You can find out who has Technology Integration Partner Program access in the *My colleagues tool*. Contact *partner-services@axis.com* to add or remove colleagues with access to the ACAP Service Portal.

5.3 Accept or deny unsigned ACAPs

Note

This feature requires ACAP version 3.2 (firmware version 10.2).

You can use a toggle to configure an Axis device to accept signed packages only.

Signed packages will be mandatory in the future. In preparation for this, you can use the toggle together with the package signing service in the ACAP Service Portal, to test the revised workflow.

The signature verification is performed during the ACAP application installation.

Important

- Any unsigned application that was installed while the toggle was set to accept unsigned packages, will still be installed after the toggle is set to **not** accept unsigned packages.
- Only use the toggle for testing purposes.

5.3.1 Code examples

Deny unsigned packages

```
gdbus call --system --dest com.axis.AcapManager1 --object-path /com/axis/AcapManager1
--method com.axis.AcapManager1.SetAllowUnsigned "false"
```

Allow unsigned packages

```
gdbus call --system --dest com.axis.AcapManager1 --object-path /com/axis/AcapManager1
--method com.axis.AcapManager1.SetAllowUnsigned "true"
```

Read the current configuration

```
gdbus call --system --dest com.axis.AcapManager1 --object-path /com/axis/AcapManager1
--method com.axis.AcapManager1.GetAllowUnsigned
```

6 API

The SDK provides the following APIs:

Video and audio APIs

- *Audio API*: Axaudio
- *Video capture API*: VdoStream
- *Machine learning API on page 31*: Larod
- *Overlay API*: Axoverlay
- *Open standard APIs*

Application support APIs

- *Parameter API*: AxParameter
- *HTTP API*: AxHTTP
- *Event API*: Axevent
- *License API*: LicenseKey

Camera feature APIs

- *Edge storage API*: AxStorage
- *Serial port API*: AxSerialPort
- *Pan tilt zoom API*: AxPTZ

Go to the *ACAP API documentation* for detailed functional descriptions and examples of the APIs.

6.1 API versions

The table below shows API and firmware version compatibility.

API version	Available from firmware version
API 3.0	9.70
API 3.1	9.80
API 3.2	10.2
API 3.3	10.5

6.2 Audio API

Go to the *ACAP API documentation* for detailed functional descriptions and examples of this API.

The Axaudio API allows the application to:

- retrieve compressed or uncompressed audio in different formats, from the camera.
- send uncompressed audio to cameras with audio output.
- configure sample rates and bitrates for some compressed formats

The API supports several get methods that returns supported formats for a camera or other audio product.

6.2.1 Compatibility

The API supports products with the following chips:

- ARTPEC-7
- ARTPEC-6
- Ambarella S5L
- Ambarella S5
- Ambarella S2L

6.2.2 Version history

This API was introduced in API version earlier than 3.0.

6.3 Video capture API

Go to the *ACAP API documentation* for detailed functional descriptions and examples of this API.

The VdoStream API provides:

- video and image stream
- video and image capture
- video and image configuration

6.3.1 Compatibility

The API supports products with the following chips:

- ARTPEC-7
- ARTPEC-6
- Ambarella S5L
- Ambarella S5
- Ambarella S3L
- Ambarella S2L

6.3.2 Version history

The VdoStream API was introduced in API version 3.0.

6.3.3 Code Examples

6.3.3.1 Capture video stream

This code example on *GitHub* starts a vdo stream and then illustrates how to continuously capture frames from the vdo service, access the received buffer contents as well as the frame metadata.

6.3.3.2 libvdo and larod combined

This code example on *GitHub* loads an image classification model to larod and then uses vdo to fetch frames of size WIDTH x HEIGHT in yuv format which are converted to interleaved rgb format and then sent to larod for inference on MODEL.

6.4 Machine learning API

Go to the *ACAP API documentation* for detailed functional descriptions and examples of this API.

liblarod lets you communicate with the Larod machine learning service, and use its features. The Machine learning API can be used for deep learning applications.

6.4.1 Compatibility

The API supports products with the following chips:

- ARTPEC-7
- Ambarella S5L

For products with a DLPU (Deep Learning Processing Unit), inference runs on the DLPU otherwise it runs on the CPU.

6.4.2 Version history

Larod API version	ACAP API version	What's new
1.0	3.1	Larod API introduced
2.0	3.3	• *Preprocessing functionality* • Handles TFLite when using firmware version 10.6 and later.

Note

We recommend using Larod version 2.0. However you can still use Larod version 1.0. To use Larod 1.0, define `LAROD_API_VERSION_1` in your application, see **Backward compatibility** in *Introduction to larod for app developers* for more information.

6.4.3 Code Examples

6.4.3.1 Extract and analyze output

This code example on *GitHub* connects to larod and loads a model, runs inference on it and then finally deletes the loaded model from larod.

6.4.3.2 libvdo and larod combined

This code example on *GitHub* loads an image classification model to larod and then uses vdo to fetch frames of size WIDTH x HEIGHT in yuv format which are converted to interleaved rgb format and then sent to larod for inference on MODEL.

6.5 Overlay API

Go to the *ACAP API documentation* for detailed functional descriptions and examples of this API.

The Axoverlay API is a helper library that enables an ACAP to draw overlays in selected video streams. It has built-in support for Cairo as rendering API, as well as an open backend for any other custom rendering.

6.5.1 Compatibility

The API supports products with the following chips:

- ARTPEC-7
- ARTPEC-6

6.5.2 Version history

The Axoverlay API was introduced in API version 3.0.

6.5.3 Code Examples

6.5.3.1 Draw plain boxes and text

This code example on *GitHub* shows how to draw plain boxes and text as overlays in a stream with the Axoverlay API.

6.6 Open standard APIs

Name	Description	Use cases	Compatibility	Documentation
Cairo 1.16.0	• Open-source rendering library for 2D vector graphics. • For use with Ax_Overlay.	• Rendering texts for internal text overlays. • Rendering lines, bounding boxes or any vector shapes. • Loading SVG.	• ARTPEC-7 • ARTPEC-6 • Ambarella S5L • Ambarella S5 • Ambarella S3L • Ambarella S2L	See *Cairo documentation*
OpenGL ES 2.0	Accelerate graphics rendering with GPU.	• Rendering mask for Axis Live Privacy Shield. • Rendering GUI for door camera with display information as PTZ or a street name.	ARTPEC-7	See *OpenGL ES documentation*

Name	Description	Use cases	Compatibility	Documentation
OpenCL 1.2	Accelerate parallel compute with GPU.	• Color conversion (e.g. as pre-processing for DL and analytics. • Audio filter. • Accelerates OpenCV.	ARTPEC-7 Note GPU is disabled on some devices due to memory limitations.	See *OpenCL documentation*
OpenVX 1.0	Accelerate computer vision with GPU.	• perspective transform (e.g. as pre-processing for DL and analytics). • Non real-time DL.	ARTPEC-7	See *OpenVX documentation*

6.6.1 Version history

The open standard APIs were introduced in API version 3.0.

6.7 Parameter API

Go to the *ACAP API documentation* for detailed functional descriptions and examples of this API.

The AxParameter API allows the application to save data and application settings so that they are not lost during a restart or firmware upgrade of the Axis product.

AxParameter makes it possible to define callback functions that perform specific actions if a parameter is updated, for example if the user updates a parameter using the Axis product's web pages.

Note

Data that's only used and altered by the application should not be handled by AxParameter. Such data can instead be handled by, for example, GLib GKeyFile.

Parameters specific to the application can be pre-configured when the application is created or be added in runtime.

6.7.1 Compatibility

The API supports products with the following chips:

- ARTPEC-7
- ARTPEC-6
- Ambarella S5L
- Ambarella S5
- Ambarella S3L
- Ambarella S2L

6.7.2 Version history

This API was introduced in API version earlier than 3.0.

6.8 HTTP API

Go to the *ACAP API documentation* for detailed functional descriptions and examples of this API.

The AxHTTP API allows the application to act as a CGI. This is done through a socket transfer to the ACAP application. This is a useful way to provide a configuration API, or updated information from the ACAP application. The information or configuration can then be accessed in ACA through a page provided by the ACAP application.

6.8.1 Compatibility

The API supports products with the following chips:

- ARTPEC-7
- ARTPEC-6
- Ambarella S5L
- Ambarella S5
- Ambarella S3L
- Ambarella S2L

6.8.2 Version history

This API was introduced in API version earlier than 3.0.

6.9 Event API

Go to the *ACAP API documentation* for detailed functional descriptions and examples of this API.

The Axevent API provides:

- an interface to the event system found in Axis products.
- applications with a mechanism for sending and receiving events.

An application can both send and receive events.

Event types

Stateless (Pulse) – An event that indicates that something has occurred. Typically used to trigger some action rule.

Stateful (State) – An event with an active and inactive state. Typically used in action rules like "record while active".

Data (Application Data) – An event that includes data that needs to be processed by the consuming application such as transaction data, license plate or other dynamic data. A data event is normally not used to trigger generic action rules.

Supported Namespaces

When declaring events it is required to set a namespace. Following are the supported namespaces:

tnsaxis – Axis namespace to use with Axis events

tns1 – ONVIF namespace to use with ONVIF events

6.9.1 Compatibility

The API supports products with the following chips:

- ARTPEC-7

- ARTPEC-6
- Ambarella S5L
- Ambarella S5
- Ambarella S3L
- Ambarella S2L

6.9.2 Version history

Event API version	ACAP API version	What's new
1.0	ACAP 2	Event API introduced
1.1	3.2	New functions: • ax_event_new2 • ax_event_get_time_stamp2

Note

- **ax_event_new** and **ax_event_get_time_stamp** functions are marked as deprecated from ACAP API version 3.2 because they use *GTimeVal*, which is deprecated in glib. Use **ax_event_new2** and **ax_event_get_time_stamp2** instead.
- To prevent installation of an application (using the Event API) on unsupported firmware (version 10.2 and earlier), set `REQEMBDEVVERSION` to `2.18` (or higher) in **package.conf** or **manifest.json**.

6.9.3 Code Examples

6.9.3.1 Subscribe to and send event

This code example on *GitHub* illustrates both how to subscribe to different events and how to send an event.

6.10 Edge storage API

Go to the *ACAP API documentation* for detailed functional descriptions and examples of this API.

The AxStorage API allows the application to save and retrieve data on mounted storage devices such as SD cards and NAS (Network Attached Storage) units. An application can only modify its own files on the storage device.

6.10.1 Compatibility

The API supports products with the following chips:

- ARTPEC-7
- ARTPEC-6
- Ambarella S5L
- Ambarella S5
- Ambarella S3L
- Ambarella S2L

6.10.2 Version history

This API was introduced in API version earlier than 3.0.

6.11 License API

Go to the *ACAP API documentation* for detailed functional descriptions and examples of this API.

Use the LicenseKey API to validate an application license key.

A license key is a signed file, generated for a specific device ID and application ID. The ACAP Service Portal maintains both license keys and application IDs.

6.11.1 Compatibility

The API supports products with the following chips:

- ARTPEC-7
- ARTPEC-6
- Ambarella S5L
- Ambarella S5
- Ambarella S3L
- Ambarella S2L

Note

For ACAPs built using **manifest.json**, license key handling is supported from firmware version 10.6 and later.

6.11.2 Version history

This API was introduced in API version earlier than 3.0.

6.11.3 Code Examples

6.11.3.1 Status of a license key

This code example on *GitHub* shows how to check the status of a license key.

6.12 Serial port API

Go to the *ACAP API documentation* for detailed functional descriptions and examples of this API.

The AxSerialPort API allows the application to configure and control the external serial port on selected Axis products.

6.12.1 Compatibility

The API is product dependent since not all Axis products are equipped with a serial port.

The API supports the RS-232, RS-422 and RS-485 standard.

The API supports products with the following chips:

- ARTPEC-7

6.12.2 Version history

This API was introduced in API version earlier than 3.0.

6.13 Pan tilt zoom API

Go to the *ACAP API documentation* for detailed functional descriptions and examples of this API.

The AxPTZ API allows the application to control the camera's pan, tilt and zoom movements, to interact with the PTZ control queue and to create and delete preset positions.

Important

The AXPTZ library is designed to be used with the GLib library. An application using the AXPTZ library must have a running GMainLoop.

Note

The AXPTZ library has an internal reference counter. If using the pattern: create library, create library, ..., call function, call function, ..., destroy library, destroy library, ..., the library will be thread safe. It is recommended to call the create and destroy functions once in the main thread of the application.

6.13.1 Compatibility

The API supports products with the following chips:

- ARTPEC-7
- ARTPEC-6
- Ambarella S5L
- Ambarella S3L
- Ambarella S2L

6.13.2 Version history

This API was introduced in API version earlier than 3.0.

7 What's new in ACAP SDK

7.1 What's new in release 3.4.2

Library or tool updates

- Changed the GCC directory search option format from `-L dir` to `-Ldir` in the SDK compiler variables in ACAP build tools. See *GCC documentation* for more information.
- Added a symbolic link that points `python` to `python3`. In ACAP SDK 3.4 and earlier, python3 is installed but some programs and scripts rely on the `python` command.

7.2 What's new in release 3.4.0

New features

- All application defines and configurations fully supported in tools for conversion to manifest based ACAP application.
- Build command updated to support reproducible builds. See *Reproducible builds on page 24* for more information.

Library or tool updates

- glibc 2.32

New APIs

- **larod 2.0** - Handles TFLite when using firmware version 10.6 and later.

7.3 What's new in release 3.3.0

New features

- **Support for manifest.json** - used for application defines and configurations such as app name and version. **manifest.json** is a JSON based manifest file enabling schema validation for error handling. The SDK includes a tool for converting package.conf to a manifest file. We recommend using the manifest now to future proof your application. See *Manifest file on page 16* for more information.

New APIs

- **larod 2.0** - New version of machine learning API with support for hardware accelerated image pre-processing operations crop, scale and color-space conversion.

7.4 What's new in release 3.2.0

New features

- A toggle that controls loading of signed ACAPs, see *Accept or deny unsigned ACAPs on page 27*.

Library or tool updates

- gcc updated to version 9.3.

Other changes

- From this release on, the ACAP SDK is available on *DockerHub* only.

Deprecated APIs

- Capture is removed from SDK and firmware after the next firmware LTS. Use *Video capture API on page 30* instead.

7.5 What's new in release 3.1.0

Library or tool updates

- gcc updated to version 9.2.

Other changes

- From this release, the ACAP toolchain and ACAP API are available as container images on *DockerHub*.

New APIs

- *Machine learning API on page 31*

7.6 What's new in release 3.0.1

Library or tool updates

- gcc updated to version 8.3.

Other changes

- ACAP SDK not available for the mips architecture from this version.

New APIs

- *Overlay API on page 31*
- *Open standard APIs on page 32*
- *Video capture API on page 30*

8 Licenses

8.1 Third party software license

You can find the ACAP SDK open source licenses and copyleft source code *here*.

8.2 End user license agreement

By downloading AXIS Embedded Development SDK, you agree to the terms in the *license agreement*.

9 Help us improve this guide

Please give us your feedback on this guide to help us improve the overall experience for you as a developer. Help us by answering the questions in *this survey*. We appreciate it!

SDK
ACAP Developer Guide - Version 3.1.0 - 3.4.2
Ver. M23.2
Date: May 2022
Part No.