TUTORIAL

Developing for ACAP

A Tutorial

Rev 1.0

Table of contents

Introduction

1.1 Overview

AXIS Camera Application Platform (ACAP) is a platform providing the means to install and run services (applications) in an Axis network video product, thus extending its capabilities. ACAP applications can be likened to smartphone apps, although these are mainly aimed at the end-user and not at system integration, as ACAP applications are.

Most applications focus on Axis network cameras, but many ACAP applications can also be installed on Axis video encoders, where relevant. In this document, “camera” is taken to signify a camera or a video encoder.

By definition, an Axis ACAP application runs at “the edge”, i.e. at the edge of the network video system, in the Axis product itself and not on a central server. Edge-based applications provide various benefits for the different parties involved:

For solution providers:

- The ability to provide a system service even if there is no other access to servers or applications
- Simplified deployment and tuning of client projects - provide a unique customized solution by combining the right cameras, the right ACAP application(s) and the right VMS solution.

For end-users:

- Reduced storage and bandwidth consumption – video is processed and stored locally
- Reduced system costs and complexity – separate servers not required
- Scalable deployment – less traffic and hardware make it easy to scale up the system with more cameras and more ACAP applications.

Depending on the use case, some services may be better suited for deployment on a server or a client. An ACAP application will always incur an overhead, although whether this will be more or less than for an equivalent server-based solution is difficult to predict.

1.2 ACAP SDK

The AXIS Embedded Development SDK (Software Development Kit) provided by Axis can be used for porting existing server applications, or as an aid in the development of new ones. The operating system in Axis network video products is Linux, and ACAP applications can be viewed as Linux applications installed on a Linux PC. The performance of an Axis network video product is, of course, not equivalent to that of a PC, but it is still sufficient for many types of service.

1.3 Application Interfaces

Most ACAP applications provide a web user-interface that is accessible via the product’s built-in web server. This interface is typically used for configuration, administration and maintenance. In most cases, an application will use an API (Application Programming Interface) in the camera, thus enabling integration with other servers, applications and systems.

1.4 Types of Application

From a system integration perspective there are basically three different types of application:

- Stand-alone applications: These are configured and run independently in the video product. This type of application can have its own user interface, or it can be integrated into systems using other standard interfaces (e.g. HTTP, FTP, RTSP, SMTP, etc.).

- VMS/Event: These applications produce one or more events intended to trigger action rules in a VMS, which typically are video analytic detectors. Examples of action rules are triggering of recordings and sending of alerts to operators.
- Proprietary components: A server-based application can install an ACAP application in order to enhance overall system performance. These are not intended for integration with a generic VMS or other such systems.

1.5 Distribution

To protect business for ACAP developers, applications can be protected by a license code, which will require a key to enable installation in the video product. Applications and license codes are distributed through the vendor’s own channels. Please see section 0 for more on this.

1.6 Hardware Platforms

The ARTPEC[1] chip was developed solely for Axis network video products, and consists of:

- Image quality processor – delivers outstanding image quality
- Video encoder – delivers video in multiple (parallel) MJPEG and H.264 streams
- CPU – General purpose CPU for hosting ACAP applications

The ARTPEC versions relevant to ACAP development are listed below. To see which chip a particular product family contains, please see http://www.axis.com/partner_pages/acap.php?t=cv&mid=2849

The currently available chip versions are as follows:

- ARTPEC-4 – A Mips-based chip with superior processor power and fast image analytics. Four times more powerful than ARTPEC-3 (the now-obsolete precursor) in terms of ACAP applications, but also a major improvement in terms of image quality.
- ARTPEC-5 – The latest and most powerful chip to date. Based on dual-core multiprocessor architecture, it features a larger cache size and higher memory throughput. More processing power and performance provide increased capabilities for intelligent video processing and video analytics applications.
- Ambarella A5S – A low-end processor on which it is possible to run analytics, but which is also more limited than the ARTPEC chips described above, especially as concerns the limited number of parallel video streams. This processor is Arm-based.
- Ambarella S2 – A high-performance chip used in Axis 4K video products. Arm-based.

1.7 Performance & Memory

Camera performance may become an issue if the ACAP application performs video analytics that require a lot of CPU power. For analytic applications, algorithms should be optimized to match the available CPU power and use-case. Often the camera will be used in a single-purpose solution, but if a network video product with video analytics is used in a multi-purpose solution, then performance should always be validated prior to deployment.

In camera firmware 5.60 and higher it is possible to install and run multiple ACAP applications, although Axis does not recommend running more than one video analytic application in a single product. Running other, non-analytic applications that only require limited CPU power is not a problem, even if an analytics application is already running. It is also possible to install multiple applications in earlier firmware versions, but only one application can be run at a time in these versions.

The amount of memory available in the video product will also influence the number of ACAP applications it is possible to install.

[1] ARTPEC® is a registered trademark of Axis Communications AB.

2 Getting Started

2.1 Prerequisite Skills and Knowledge

Applications for the AXIS Camera Application Platform (ACAP) are developed in C or C++, in a Unix/Linux environment, using various publicly available tools, as well as tools made available by Axis.

Development of applications using the AXIS Embedded Development SDK (Software Development Kit) requires the following skills and knowledge:

- Experience of GNU/Linux® or similar platforms
- Experience of developing for embedded platforms
- Knowledge of the C/C++ programming language and related libraries
- Basic knowledge of HTML and Javascript
- Experience of common development tools such as gcc and make
- Knowledge of `Glib`[2]. The version used in the SDK is 2.36 and is the minimum required API version to use when writing applications.

Axis cannot provide training or documentation on any of the above – it is up to you to acquire these skills yourself.

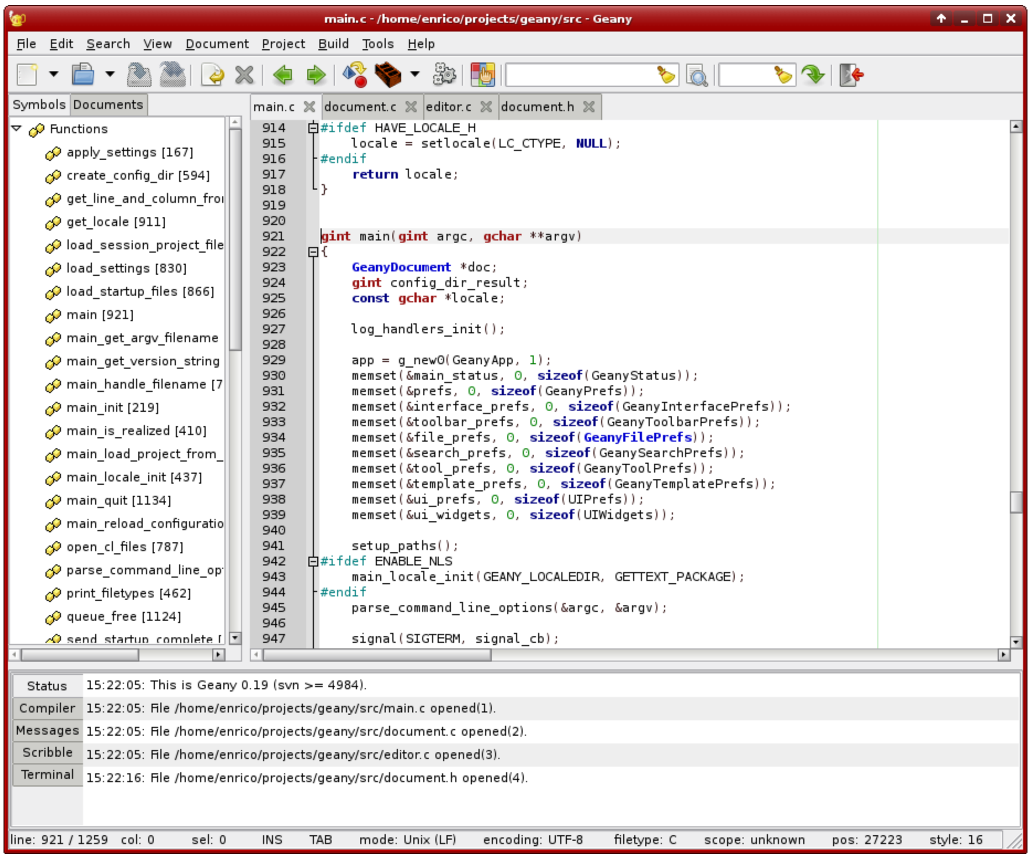
2.2 Technical Prerequisites – Hardware and Software

Developing applications with the SDK requires the following components and tools:

- A workstation running a Linux-based operating system. Axis recommends Debian or Ubuntu for ease of use. Whichever flavor you use, it must be capable of installing `.deb` packages. It is also possible to use a virtual (Linux) machine installed on e.g. Windows, in Oracle VM VirtualBox[3] or VMware Player[4]. A graphical user interface is not strictly necessary for development purposes, but may prove useful.
- The AXIS Embedded Development SDK. The latest version (2.0) contains the latest features and the most functionality, and is the recommended version. It includes the compilers for ARTPEC-4/5-based products (MIPS), and Ambarella-A5S-based products (ARM). For use in Axis products running firmware 5.60 or later.
- An Axis network video product with support for ACAP applications (FW 5.60 and later). This is available for most Axis products, but is also firmware-dependent. To check the firmware for your product, please see the page at http://www.axis.com/techsup/firmware.php
- The `Bash` shell for your version of Linux (other shells may also work).
- The `curl` package, for running the SDK script `eap-install.sh.`
- The autoconf and libtool packages - for compiling the RAPP library (low-level image processing operations).

[2] `GLib` is developed and maintained by the GNOME Project and is released under the GNU Library General Public License (GNU LGPL).
[3] Oracle and Java are registered trademarks of Oracle and/or its affiliates.
[4] VMware Player™ is a registered trademark of VMware.

- An editing tool for developing your code. One example of such a tool is Geany, which is free to use, provides syntax color highlighting, and also has a built-in terminal and buttons for build scripts.



```
Functions
    apply_settings [167]
    create_config_dir [594]
    get_line_and_column_fro
    get_locale [911]
    load_session_project_file
    load_settings [830]
    load_startup_files [866]
    main [921]
    main_get_argv_filename
    main_get_version_string
    main_handle_filename [7
    main_init [219]
    main_is_realized [410]
    main_load_project_from_
    main_locale_init [437]
    main_quit [1134]
    main_reload_configuratio
    open_cl_files [787]
    parse_command_line_op
    print_filetypes [462]
    queue_free [1124]
    send_startup_complete [
```

```
#ifdef HAVE_LOCALE_H
    locale = setlocale(LC_CTYPE, NULL);
#endif
    return locale;
}


gint main(gint argc, gchar **argv)
{
    GeanyDocument *doc;
    gint config_dir_result;
    const gchar *locale;

    log_handlers_init();

    app = g_new0(GeanyApp, 1);
    memset(&main_status, 0, sizeof(GeanyStatus));
    memset(&prefs, 0, sizeof(GeanyPrefs));
    memset(&interface_prefs, 0, sizeof(GeanyInterfacePrefs));
    memset(&toolbar_prefs, 0, sizeof(GeanyToolbarPrefs));
    memset(&file_prefs, 0, sizeof(GeanyFilePrefs));
    memset(&search_prefs, 0, sizeof(GeanySearchPrefs));
    memset(&tool_prefs, 0, sizeof(GeanyToolPrefs));
    memset(&template_prefs, 0, sizeof(GeanyTemplatePrefs));
    memset(&ui_prefs, 0, sizeof(UIPrefs));
    memset(&ui_widgets, 0, sizeof(UIWidgets));

    setup_paths();
#ifdef ENABLE_NLS
    main_locale_init(GEANY_LOCALEDIR, GETTEXT_PACKAGE);
#endif
    parse_command_line_options(&argc, &argv);

    signal(SIGTERM, signal_cb);
```

```
15:22:05: This is Geany 0.19 (svn >= 4984).
15:22:05: File /home/enrico/projects/geany/src/main.c opened(1).
15:22:05: File /home/enrico/projects/geany/src/document.c opened(2).
15:22:05: File /home/enrico/projects/geany/src/editor.c opened(3).
15:22:16: File /home/enrico/projects/geany/src/document.h opened(4).
```

```
line: 921 / 1259   col: 0   sel: 0   INS   TAB   mode: Unix (LF)   encoding: UTF-8   filetype: C   scope: unknown   pos: 27223   style: 16
```

2.3 Setting Up and installing on a Linux System

The following instructions assume that you have a Linux system up and running, with root privileges as required, with Internet access, and with an Axis network video product accessible on the same network as the Linux machine.

1. Start by checking that the `bash` shell is installed. Run the following at the terminal prompt:
   `user@host$ echo $SHELL`

   This will return the current shell, e.g. `user@host$ /bin/dash`

   To temporarily change to `bash`, simply type `user@host$ bash`

   To change the login shell to `bash,` type: `user@host$ chsh [password] /bin/bash`

   The next time you log out and in again, the shell will be changed permanently to `bash`.

2. Download the SDK from the Axis Partner Pages. Do this from the command line or via a browser in the graphical user interface.

3. Unpack the `tar.gz` file using the following in the terminal prompt:
   `user@host$ tar -xvzf AXIS_Embedded_Development_SDK_2_0.tar.gz`

The files should be unpacked to a folder in the user’s home directory, or there may be problems when accessing them.

After unpacking, the files will be arranged as below:

```
user@host$ cd axis/emb-app-sdk_2_0
user@host$ ls -1 --file-type
    apps/           The SDK example applications.
    docs/           The SDK documentation.
    init_env        The SDK environment file.
    libs/           Libraries used by the SDK interfaces.
    licenses.txt    The licenses file.
    RELEASENOTES    SDK release notes.
    target/         The SDK header files and the precompiled binaries.
    tools/          The application package creation tools.
```

❗ The unpack operation can also be done using the file manager in the graphical user interface.

4. From anywhere in the system, install the `curl` package, to be able to run the SDK script `eap-install.sh`. This requires root privileges to install.

   ```
   user@host$ sudo apt-get install curl
   ```

5. The `autoconf` and `libtool` packages are required to compile the RAPP library (for image processing). These require root privileges to install.

   ```
   user@host$ sudo apt-get install autoconf
   user@host$ sudo apt-get install libtool
   ```

You should now have a Linux system ready and capable of developing and testing applications for ACAP.

2.4 Creating and Installing a Test Application Package

To quickly get started with the AXIS Embedded Development SDK, various example applications are provided in the /apps directory in the SDK root directory. The analyze application provides a basic method of configuring a motion detection window and is an example of a Linux daemon that uses several of the interfaces[5] provided by the SDK.

1. Start by sourcing the file init_env in the SDK root directory.

```
user@host$ pwd && ls
/home/user/axis/emb-app-sdk_2_0
apps docs  init_env  libs  licenses.txt  RELEASENOTES  target  tools
user@host$ source init_env
```

Sourcing init_env will set various variables, check some tool locations and also append a few folders to the path. This is a required step to be able to use the SDK tools, and must be repeated for each new terminal opened.

2. Change to the directory containing the example application:

```
user@host$ cd apps/analyse
```

This directory contains the following files:

Makefile	The application Makefile.
analyze.h	Header file.
analyze_error.c	Source file
analyze_error.h	Header file.
analyze_util.c	Source file.
analyze_util.h	Header file.
declarations/	Directory containing event declaration.
http_cgi_paths.txt	Configuration file containing CGI-paths.
main.c	Source file.
motion_detection.c	Source file.
motion_detection.h	Header file.
package.conf	The application package configuration.
param.conf	The application parameters definition.
storage.c	Source file.
storage.h	Header file.

3. To compile the application, run the script create-package.sh, which does the following:
   - Creates the file .target-makefrag
   - Executes make in the current directory
   - Checks that the file package.conf exists and that is contains no errors
   - If required, asks for the parameters and creates package.conf
   - Creates and empties the file param.conf
   - Creates an eap package with the application binary, HTML files (if used) and configuration files.

[5] Descriptions of the available SDK interfaces can be found in the /docs directory.

When executed with no options, this command will create one package for each CPU (processor) supported by the SDK version.

```
user@host$ create-package.sh
```

To create a package for a specific CPU only, use e.g.:

```
user@host$ create-package.sh artpec-4
```

or

```
user@host$ create-package.sh mipsisa32r2el
```

You have now created an eap package!

3 Developing Your Own Applications

The basic path to follow when developing applications for ACAP is as follows:

- Gather and install all required components
- Set up the development environment
- Obtain an application ID[6] from the Axis Partner Helpdesk
- Develop the application
- Design and code the user interface (if required)
- Run `makefile`
- Compile package and install in video product
- Test and debug
- License and distribute

3.1 Available Libraries & Functions

- Capture – to grab a stream of JPEG images, or Raw Y800 images for analysis purposes.
- Fixmath – for floating point calculations with integer, fast library, fixmath library 32-bit.

3.2 Code Examples

For documented and tested code examples, please see the files available in the downloaded AXIS Embedded Development SDK.

3.3 Makefile

The Makefile provides the instructions compiling and linking a program. C/C++ source files must be recompiled every time they are changed. If a header file is changed, then each C/C++ source file that includes that header file must also be recompiled, to be on the safe side.

Each compile produces an object file that corresponds to the source file. If the source file is recompiled, all the object files (new or saved from previous compilations) must be linked together to produce a new executable program. These instructions with their dependencies are specified in the Makefile. If none of the prerequisite files have been changed since the previous compilation, then no actions occur. For large software projects, Makefiles can significantly reduce compile times when only a few source files have been changed.

[6] This is required if planning to use the copy protection tool.

3.3.1 Example Makefile

```
# Example from the axparameter application, located in the apps-folder of the SDK.
# Standard libraries supported. Does not normally need editing.
AXIS_USABLE_LIBS = UCLIBC GLIBC
# Includes all the rules, such as the compiler to use, and any special flags
required # for building applications for the target platform. This line must always
be in the
# Makefile. AXIS_TOP_DIR is the path to your SDK installation and is automatically
set # when running source ./init_env as described previously in this tutorial.
include $(AXIS_TOP_DIR)/tools/build/rules/common.mak
# The rest of this file uses standard GNU Make format. Manual available at
# www.gnu.org. Many tutorials available online.
# This sets the variable PROGS. Typically most useful to use parameters such as this
# when there are multiple applications and sources built from the same Makefile.
This # allows using the same rule for subsequent application builds.
PROGS      = parameterApp
# Additional flags for the compiler. Used to e.g. control the type of warnings
# required. See the compiler manual for more information.
CFLAGS    += -Wall -g -O2
# pkg-config is a standard helper tool used when compiling applications and
libraries, # for setting the correct flags for the compiler. Instead of hard-coding
the paths to # find the used libraries, this application finds and sets the correct
values
# automatically.
PKGS = glib-2.0 gio-2.0 axparameter
CFLAGS += $(shell PKG_CONFIG_PATH=$(PKG_CONFIG_LIBDIR) pkg-config --cflags $(PKGS))
LDLIBS += $(shell PKG_CONFIG_PATH=$(PKG_CONFIG_LIBDIR) pkg-config --libs $(PKGS))

# Specifies the sources to be built.
SRCS       = axparameterexample.c
OBJS       = $(SRCS:.c=.o)

# Sections below are the targets, i.e. when running eap-create, this ends by running
# "make all", which in turn will run the $(PROGS) target that translates to
# "parameterApp" as set in previous sections.
# The "parameterApp" target then builds and links the sources and outputs the
# executable file parameterApp.
all: $(PROGS)
$(PROGS): $(OBJS)
            $(CC) $(LDFLAGS) $^ $(LIBS) $(LDLIBS) -o $@

# The clean target should be run to switch platforms, i.e. to create binaries for
# other architectures. When running eap-create this is handled automatically. When
# writing own Makefiles, ensure that all object files and the resulting executable
# file are removed from this target.
clean:
            rm -f $(PROGS) *.o
```

4 Debugging

4.1 What is Debugging?

Debugging is the systematic process of locating, isolating and fixing errors (bugs) in computer program code, which can be designed to run on a computer or be embedded in a hardware device. Debugging is more difficult when multiple systems are interacting with each other, as changes to one system may cause bugs in another.

Debugging tools (debuggers) assist in identifying coding errors at various stages of code development, with some programming languages including functions for checking the code as it is written.

Debugging tools can be command line-based or they may provide a graphical user interface. Choose whichever you feel most comfortable with, and learn how it works. A brief overview of how to get started debugging with `gdb` is provided below, but there are many other tools and tutorials available on the Internet. One simple and quick way to debug is to write messages to the camera's log file.

4.2 Debugging with gdb

Debugging an ACAP application with `gdb` involves two separate components:

- The client-side `gdb`, which is run on the development PC
- The `gdbserver`, which is run in the video product

The required `gdbserver` files are included in Axis standard firmware as of version 5.80, and are also included in some service pack releases. To obtain these files separately, download Axis ACAP debug tools from the Axis Partner Pages, and then follow the instructions in 4.3.

If your video product uses firmware 5.80 or later, you can skip to section 4.4.

4.3 Separate Installation of the Debugger

1. The downloaded debug file should first be unpacked to the development PC:
   ```
   user@host$ tar –xf Axis_ACAP_debug_tools_<File_release_date>
   ```
   The unpacked file creates the folder `Axis_acap_debug_tools`, with two sub-folders:

   `host/` – contains the files required by `gdb` on the development PC. See 4.4.
   `target/`– contains the `gdbserver` files for upload to the video product.

`gdbserver_mips32`	– for MIPS/ ARTPEC -4/5 products
`gdbserver_arm`	– for ARM/Ambarella-A5S|S2 products

2. From the `target/` folder, upload the correct `gdbserver` file for the target platform/camera, placing it in one of the camera's writable folders, `/tmp` or `/etc`.
   ```
   user@host$ cd /home/user/Downloads/Axis_acap_debug_tools/target
   user@host$ ftp
   ftp> open <enter_camera_IP>
   Connected to <camera_IP>
   220 AXIS P1354 Fixed Network Camera 5.60.1.2 (2015) ready.
   Name (camera_IP:user)<enter_user>
   331 User name okay, need password.
   Password: <enter_password>
   230 User logged in, proceed.
   Remote system is UNIX.
   Using binary mode to transfer files.
   ftp> cd etc
   ```

```
250 Command successful.
ftp> put gdbserver_<enter_platform>
local: gdbserver_<platform> remote: gdbserver_<platform>
229 Entering Extended Passive Mode
150 Opening data connection.
100% ********************** 264 KiB 883.75 KiB/s 00.00 ETA
226 Transfer complete.
270460 bytes sent in 00.00 (702.61 KiB/s)
ftp>
```

3. Finally, you need to make the file executable:

```
ftp> chmod 755 gdbserver_<platform>
200 Command okay.
ftp> Quit
221 Goodbye.
user@host$
```

4.4 Debugger on the Development Host

The files contained in the `/host` folder are `.deb` files used by the development PC to connect to the `gdbserver` in the host. Setting the environment variables (`source init_env`) will automatically locate these files and make them available to the various scripts.

4.5 Cross-compile and Upload before Debugging

In the example on this page, the example application `analyze` will be debugged. The product's IP address is `<camera_ip>`. **Do not** strip the binary before uploading it to the Axis product.

1. Create the application package in the `apps/analyze` directory:
```
user@host$ make debug
user@host$ make clean
user@host$ create-package.sh
```

2. Upload the application package to the Axis product:
```
user@host$ eap-install.sh <camera_ip> <root_password> install
```

4.6 Starting gdbserver on the Axis Product

After the application is compiled and uploaded, the `gdbserver` is started on the Axis product, after logging in as root. This is done by accessing the camera via SSH. If not enabled on the product, follow these steps first to enable SSH:

1. Access the camera via a web browser and go to:
Setup > System Options > Advanced > Plain Config
2. Select the parameter group for **Network**.
3. Go to the parameter **Network SSH** and check the box for **SSH Enabled**.
4. Click the **Save** button.

You can now access the camera via SSH and enable the `gdbserver,` as in the following example for the `analyze` application:

```
user@host$ ssh <camera_ip>
root@camera# cd /usr/local/packages/analyze/
root@camera# gdbserver :1234 analyze
```

Leave this terminal window open, as `gdbserver` in the camera is now expecting a TCP connection from the development PC, to allow the application to be debugged. The example above uses port 1234 on the camera, although another (unoccupied) port can be used if required.

4.7 Using gdb on the Development PC

1. To start the debugger, run the wrapper script rungdb from the analyze directory.
   ```
   user@host$ rungdb analyze
   ```
2. Type `targ` at the `gdb` prompt to connect to the target (the Axis product):
   ```
   (gdb) targ
   0x3aaac8b0 in _start () from /home/user/...
   ```
3. Type `det` to detach from the target.
   ```
   (gdb) det
   Ending remote debugging.
   ```

The `rungdb` script uses the target IP address as defined by `AXIS_TARGET_IP` (default is 192.168.0.90), and the port as defined by `AXIS_GDB_PORT` (default 1234). The script automatically sets up the debugger for the target architecture and loads the debugging symbols for the `analyze` program.

4.7.1 Custom or Manual Setup

The environment variables can be overridden by setting custom network parameters:

```
user@host$ AXIS_TARGET_IP=<target_ip> AXIS_GDB_PORT=<debug_port> rungdb
analyze
```

For greater control, it is also possible to set up the debugger manually. Be sure to specify the same port as the one used to start `gdbserver` above.

```
user@host$ gdb-mips analyze
(gdb) set solib-absolute-prefix /dev/null
(gdb) set solib-search-path <path to SDK>/target/mipsisa32r2el-axis-linux-
gnu/lib
(gdb) directory <path-to-camera>/usr/lib/mylib        //specify where a
shared library "mylib" is located (if needed by the analyze program)
(gdb) target remote <camera_ip>:<debug_port>
Remote debugging using <camera_ip>:<debug_port>
0x3aaac8b0 in _start () from /home/user/...
```

For more on debugging with `gdb`, see e.g. https://sourceware.org/gdb/current/onlinedocs/gdb/

4.8 Common Problems

4.8.1 Scripts Not Running as Expected

Remember to run `source init_env` for each new terminal console opened.

4.8.2 Application not running fast enough

- Could be due to an excessive load on the platform. Try targeting other products built on a later version of ARTPEC.
- Possibly due to use of floating point calculations in applications, which is not supported in the hardware. If running open-source image analytics, the library probably uses floating points - be sure to compile this library without floating point support. Use the Fixmath library instead.
- RAPP can be used to speed up video analytic applications.

4.8.3 Application not showing on the Axis.com web page

Before an application can be made visible to all external users, it must be submitted to the Axis Application Database. This involves reporting the compatible Axis product family/families via the Axis

ADP helpdesk, as well as providing detailed information about your solution and a list of supported Axis products on your own website.

After this information has been reviewed and approved, it will be published on the Axis website. All applications are, however, available in the internal version of Axis Application Database, frequently used by Axis salespersons to find appropriate applications for their projects.

4.8.4 Cannot Generate licenses

- Application is not compatible or published for the target camera. The admin interface requires one camera per group to be "compatible", meaning that licenses can be generated.
- "Published" means licenses can be generated for the camera, and any user can generate trial licenses at Axis.com

4.8.5 Application ID Requested but not Received

Find the application in the Admin tool and change status from "pre-development" to "in development".

4.8.6 Build Scripts not Working

- Check for the correct shell in Linux - `bash` is required.
- Check that `curl` is installed – this is required for uploading packages to the camera.
- Packages built for incorrect CPU/platforms will not be accepted.

5 Copy Protection

Copy protection is used for the unlocking of ACAP applications in Axis network video products. License codes are generated in the Axis copy protection service tool, and are then sold via the partner/developer's own sales channels. Codes can be distributed together with the application, or supplied separately.

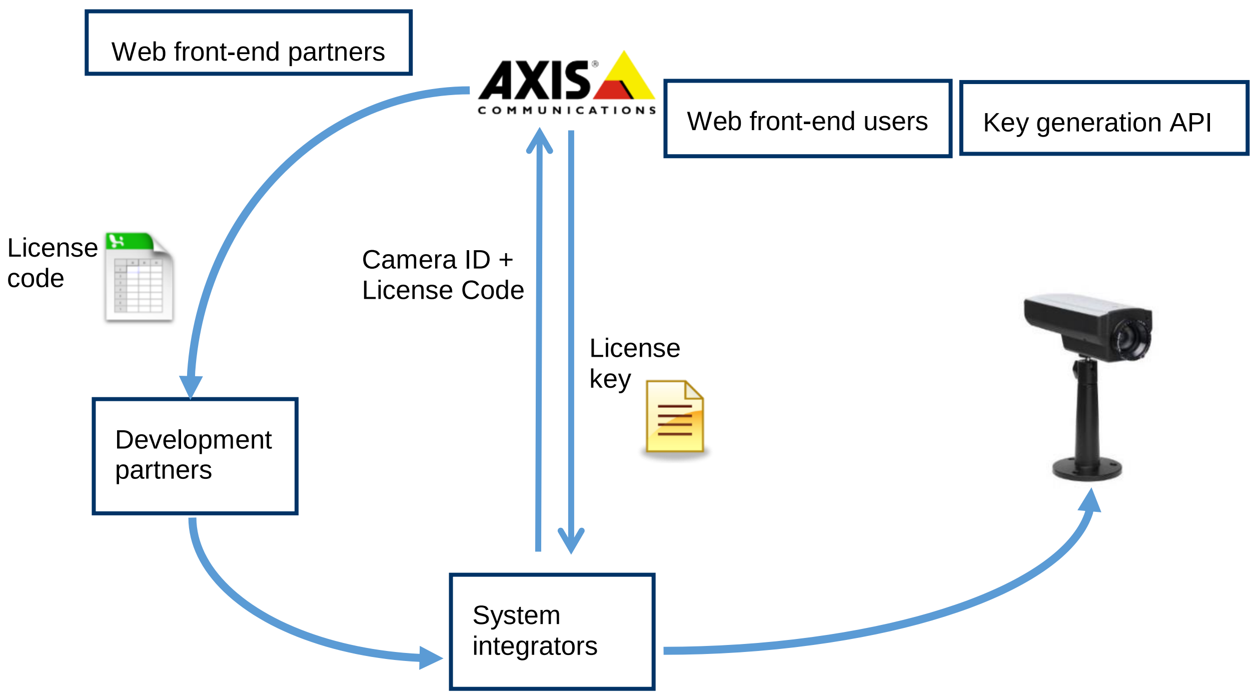
This solution also means that partners can receive valuable feedback about which product the applications was installed in, the user that installed it, etc.

5.1 How Does it Work?

The system is based on license codes and keys, the latter being installed in the network video product. The code itself is merely a random number, designed to make it that little bit harder for an attacker to bypass the protection.

Codes can be generated by the ACAP partner, and these are linked internally in the Axis system to a specific application. Axis offers a front-end both for the administration of codes and for a system integrator to generate keys.

Entering a valid code and a camera serial number into the system will generate a device-specific license key, which is then uploaded to that device in order to activate the purchased ACAP application. It is also possible to generate trial/demo keys for evaluation purposes.



A code can be made valid for a set number of installations. After this limit is reached the system will not generate further keys for that code.

5.2 Using the Copy Protection Scheme

An application ID is required for your ACAP application in order to take advantage of the copy protection scheme. This can be obtained via the Axis Partner Pages.

When developing your application, observe the following in the code:

- Always link statically to the license library, which protects against attempts to override the path.

- Call the `licensecheck` function from several places in your code, which makes it harder to replace the call with a NOP instruction.
- When using time-based licenses, make regular calls to the function to check for expiry.

5.3 Example of a Call to the Licensecheck Function

```
int checkLicense()
{
  int ret;

  /* Axis License API, this call should be substituted in an
   * application using a custom/proprietary licensing framework.
   * see customlicense.h for more information.
   * If a custom/proprietary license framework is used, set the
   * package.conf variables LICENSEPAGE and LICENSE_CHECK_ARGS.
   * e.g:
   * LICENSEPAGE='custom'
   * LICENSE_CHECK_ARGS='--license_status'
   */

  ret = licensekey_verify(PACKAGE_NAME, APP_ID, MAJOR_VERSION,
MINOR_VERSION);

  if (ret) {
    LOG("%s: License verification succeeded\n", PACKAGE_NAME);
    // Start the application
  }
  else {
    LOG("%s: License verification failed\n", PACKAGE_NAME);
   // No license or not valid, close the application
  }
  return ret;
}
```

The example above can also be found in `/apps/licensecheck`