4.5 Software Development Process

This section provides information about embedded developer resources and procedures from the Impinj reader SDK. It enables a developer to write an example application, generate the appropriate binaries, install the example application on a reader, and execute the application.

4.5.1 Tools

The Impinj reader filesystem is based on the Debian GNU/Linux distribution. The version of Debian and the version of the Linux kernel can be obtained with the shell command `uname -a`.

An embedded software developer can develop a custom application by using the Impinj reader Embedded Toolkit (ETK) available from the Impinj Support Portal. This package includes the Impinj LLRP libraries for C and C++, a sample application, a cross-compiled toolchain, and other utilities.

4.5.2 Create a Linux environment

Use of the Impinj Reader Firmware ETK requires a GNU/Linux Operating System. Create a Linux environment with your preferred method (desktop, virtual machine, Vagrant, etc.). This guide assumes the following commands have been run successfully against an Ubuntu 20.04 Linux system, at minimum:

```
$ sudo dpkg --add-architecture i386
$ sudo apt-get update --fix-missing
$ sudo apt-get install -y apt-utils
$ sudo DEBIAN_FRONTEND=noninteractive apt-get install -y build-essential
$ sudo DEBIAN_FRONTEND=noninteractive apt-get install -y cmake
$ sudo DEBIAN_FRONTEND=noninteractive apt-get install -y libc6-i386
$ sudo DEBIAN_FRONTEND=noninteractive apt-get install -y libssl-dev
$ sudo DEBIAN_FRONTEND=noninteractive apt-get install -y zlib1g:i386
```

Note: In the commands below, `vim` is used, but feel free to use any other text editor.

4.5.3 Setting up a development environment

4.5.3.1 Download the Impinj Reader Firmware ETK From the Impinj Support Portal, download the most recent Impinj Reader Firmware ETK file, for example:

```
8.1.6.0_Octane_Embedded_Development_Tools.tar.gz
```

4.5.3.2 Extract the Impinj Reader Firmware ETK Navigate to the directory where the tarballs are located. Use the following commands to extract the Impinj Reader Firmware ETK.

```
$ tar xzfv 8.1.6.0_Octane_Embedded_Development_Tools.tar.gz
$ cd 8.1.6.0_Octane_Embedded_Development_Tools
```

Now your environment is ready to compile code to run on Impinj Reader Firmware.

4.6 CAP creation

4.6.1 Overview

A CAP is a custom application that is loaded onto an Impinj RFID reader. An on-reader application is installed onto the reader via an upgrade file using the same upgrade mechanism as the Impinj reader firmware. Impinj provides a utility to convert a directory structure along with its contents into an upgrade (`.upgx`) file. This tool is called `cap_gen` and is included in the ETK. Most of `cap_gen`'s arguments are passed in a text description file. An explanation of each option is included in the ETK's example `cap_description.in` file.

Note: `cap_gen` depends on `fstool`. This tool is also included in the ETK.

Here is an example of a `cap_gen` description file:

```
# Upgrade Description File
#
# This file contains the settings used by the upgrade generation tool
# to produce an Impinj reader firmware upgrade file.
[Description]

# Version is a 4 part number in decimal with each part limited to
# 0-255. It is the version of the upgrade file to be generated.
Version = 1.2.3.4

# Valid Reader Hardware is a string designating the reader models onto
# which you can load the upgrade. The following strings are acceptable
# values for this field.
#
#  R700
#  R700V2
#  R7**
#  R***
#
# The "R***" wildcard indicates that you can load the CAP onto any
# reader, and effectively bypasses this check.
R700

# File System Layout is an value used by the reader to determine how
# the upgrade partition should be loaded to flash. Currently the only
# supported layout version is 10.
File System Layout = 10

# Input Directory is the top-level directory of the filesystem to
# create. The files under this directory will be available on the
# reader under /cust after the upgrade is loaded.
Input Directory = ./cap

# Encrypt the upgrade filesystem content. Note this will be
# decrypted when the upgrade file is processed by the reader.
# The default is 'False' (don't encrypt)
#Encrypt = True

# Should point to the cert from the key pair used to encrypt the bundle.
#Recipients = .//cap_enc.cert.pem

# The above cert.pem can be generated with:
#
# openssl req -x509 -newkey rsa:4096 -keyout cap_enc.key.pem -out \
#   cap_enc.cert.pem -sha256 -days 365
#
# Choose any passphrase; you only have to enter it in the next step. Create
# a p12 bundle:
#
# openssl pkcs12 -export -out cap_enc_key-bundle.p12 -in cap_enc.cert.pem \
#   -inkey cap_enc.key.pem
```

```
#
# Here, you must define a passphrase for the p12 bundle after entering the pem
# passphrase.
#
# Then upload bundle to the reader:
#
# curl -k -u root:impinj -X POST -F "upload=@cap_enc_key-bundle.p12" \
#   -F"password=" \
#   https://$READER_IP/api/v1/system/certificates/tls/certs
#
# Which should reply with [1] or whatever ID was assigned to the cert.
# Activate CAP encryption for that cert:
#
# curl -k -u root:impinj -X PUT -d "{"certId":1}" \
#   https://$READER_IP/api/v1/system/certificates/tls/services/cap-encryption

# Use the expanded NAND Flash partition sizes available on
# all Speedway based readers and gateways with PCBAs equal or
# greater than v5.00.
# The default is 'False' (build for all PCBAs).
#ExpandedNAND = True

# CAP Partition size can be 128MB or 256MB
# Using the 128MB size is recommended unless the extra space is specifically needed
Partition Size = 128
```

Several of these options are described in further detail below.

The `Version` field should be the version number of the on-reader application. This is the version number that RShell will report for the CAP. It is also used by the auto upgrade mechanism to determine if the CAP needs to be upgraded, details of which are covered in the ***Firmware Upgrade Reference Manual***.

`Input Directory` tells `cap_gen` the root of the file structure it should include in the CAP upgrade file.

The content of a CAP firmware upgrade file can be encrypted by setting the `Encrypt` field to `True`. This applies a symmetric block cipher to the data (directories and files) contained in the CAP upgrade file. It will automatically decrypt the content when processing the upgrade file. Impinj recommends that an OSShell password be included in the CAP to further protect the content of the encrypted CAP upgrade file.

4.6.2 Using cap_gen

For R700, the `cap_gen` utility includes extended functionality:

```
Usage: cap_gen.sh [-v] [ |  | -V]
  -v                Output verbose information
  -d file           The upgrade description file to use for settings
  -f {mkfs|fstool}  The external tool to use for filesystem generation
  -o file           The output file to write
  -i file           The image file to inspect and display metadata
  -k directory      The directory containing private key and cert for signing.
                        The directory should contain cap.key.pem and cap.cert.pem.
  -n                Generate new private key and cert for signing.
  -V                Output version information
```

`cap_gen` would typically be invoked as follows to generate an upgrade file:

```
$ cap_gen.sh -d cap_description.in -o my_cap.upg
```

To view details of an existing upgrade file:

```
$ cap_gen.sh -i my_cap.upg
Partition START
  ValidHardware[00]      : 225-255.255
  ValidHardware[01]      : 240-255.255
  ValidHardware[02]      : 250-255.255
  ValidHardware[03]      : 260-255.255
  ValidHardware[04]      : 270-255.255
  ValidHardware[05]      : 280-255.255
  ValidHardware[06]      : 290-255.255
  FormatVersion          : 3
  Flags                  : 0x0000
  FirmwareVersion        : 1.2.3.4
  FileSystemLayout       : 10
  Partition              : CAP
```

4.6.3 Compiling a sample application

To build the ETK examples using your native environment, `cd` to the appropriate directory and call the `build.sh` script.

```
[user@machine]$ pwd
//8.0.0_Octane_Embedded_Development_Tools
[user@machine]$ cd //examples/ltk-c/ && ./build.sh
[user@machine]$ cd //examples/ltk-cxx/ && ./build.sh
```

It's also possible to build and run the examples using Docker. For more information, see the file `README.md` in the ETK archive.

4.6.4 Custom application

To build the example CAP, `cd` to the `cap` directory and call `make`:

```
[user@machine]$ pwd
//8.0.0_Octane_Embedded_Development_Tools/examples
[user@machine]$ cd cap
[user@machine]$ make
```

The build produces a `cap.upgx` which can then be installed on an Impinj Reader via the web UI.

4.6.4.1 libpthread As of Impinj Reader Firmware 8.1.6, you must link the CAP and the `libpthread` binary dynamically to avoid a segfault of the CAP. In addition, anyone using the CAP must modify the `/customer/start` script with the following two lines to configure the library load path and avoid library load errors from the `example` executable.

```
LD_LIBRARY_PATH="/customer/lib/"
export LD_LIBRARY_PATH
```

4.6.4.2 Enter OSShell `ssh` into the reader. After you connect to RShell, enter the following command to get access to the Linux shell.

```
> osshell developer
```

4.6.4.3 Running the applications The following slightly edited console log shows how to run the example files on an Impinj R700 RAIN RFID reader.

```
root@impinj-14-50-de:~# cd /cust
root@impinj-14-50-de:/cust# ./docsample1_c/docsample1 127.0.0.1
INFO: Starting run 1 ================
INFO: 2 tag report entries
E280-1190-A503-1FA0-0000-0002
3008-33B2-DDD9-0140-0000-0000
INFO: Starting run 2 ================
INFO: 2 tag report entries
E280-1190-A503-1FA0-0000-0002
3008-33B2-DDD9-0140-0000-0000
INFO: Starting run 3 ================
INFO: 2 tag report entries
E280-1190-A503-1FA0-0000-0002
3008-33B2-DDD9-0140-0000-0000
INFO: Starting run 4 ================
INFO: 2 tag report entries
E280-1190-A503-1FA0-0000-0002
3008-33B2-DDD9-0140-0000-0000
INFO: Starting run 5 ================
INFO: 2 tag report entries
E280-1190-A503-1FA0-0000-0002
3008-33B2-DDD9-0140-0000-0000
INFO: Done

root@impinj-14-50-de:/cust# ./docsample1_cxx 127.0.0.1
EPC: 3008-33B2-DDD9-0140-0000-0000
EPC: E280-1190-A503-1FA0-0000-0002
[many similar lines]
INFO: Done
root@impinj-14-50-de:/cust#
```

4.7 CAP Authentication

If you create a CAP with the `cap_gen` script, the upgrade file will be automatically signed with its own certificate. You can also use your own certificate to sign the upgrade image.

Note: We will assume we are in the `examples` directory in the ETK for the rest of the document.

```
$ cd /examples
```

4.7.1 Settings

There are three settings for CAP authentication:

- `Disabled`: CAPs cannot be installed on the reader (default).
- `Open`: Unencrypted default-signed or unencrypted custom-signed CAPs can be installed.
- `Secure`: Only encrypted or unencrypted custom-signed CAPs can be installed.

These settings can be changed via the WebUI or the IoT Device Interface (REST API).

4.7.2 Generating a custom-signed CAP

To generate a custom-signed CAP, use the `-k` option in `cap_gen` to specify the directory that the certificate and key are in. This directory passed in should contain a `cap.cert.pem` and a `cap.key.pem` . The `cap.key.pem` is the key used to create the certificate.

Note: The key is only used to sign the image and will not need to be uploaded to the reader or integrated into the CAP.

The `-sha256` flag will prompt for a password and prompt again once you generate the CAP. You can specify `-nodes` instead if you want to skip this feature.

4.7.3 Generating a self-signed certificate and key

```
$ openssl req -x509 -newkey rsa:4096 -keyout cap.key.pem -out cap.cert.pem -sha256 -days 365
$ mkdir cap_signing
$ mv cap.cert.pem cap_signing
$ mv cap.key.pem cap_signing
```

4.7.4 Generating the signed upgrade image

```
$ ../cap_gen.sh -d cap/cap_description.in -k cap_signing -o my_cap.upgx
```

The CAP will now be signed with the private key in the directory specified with `-k`.

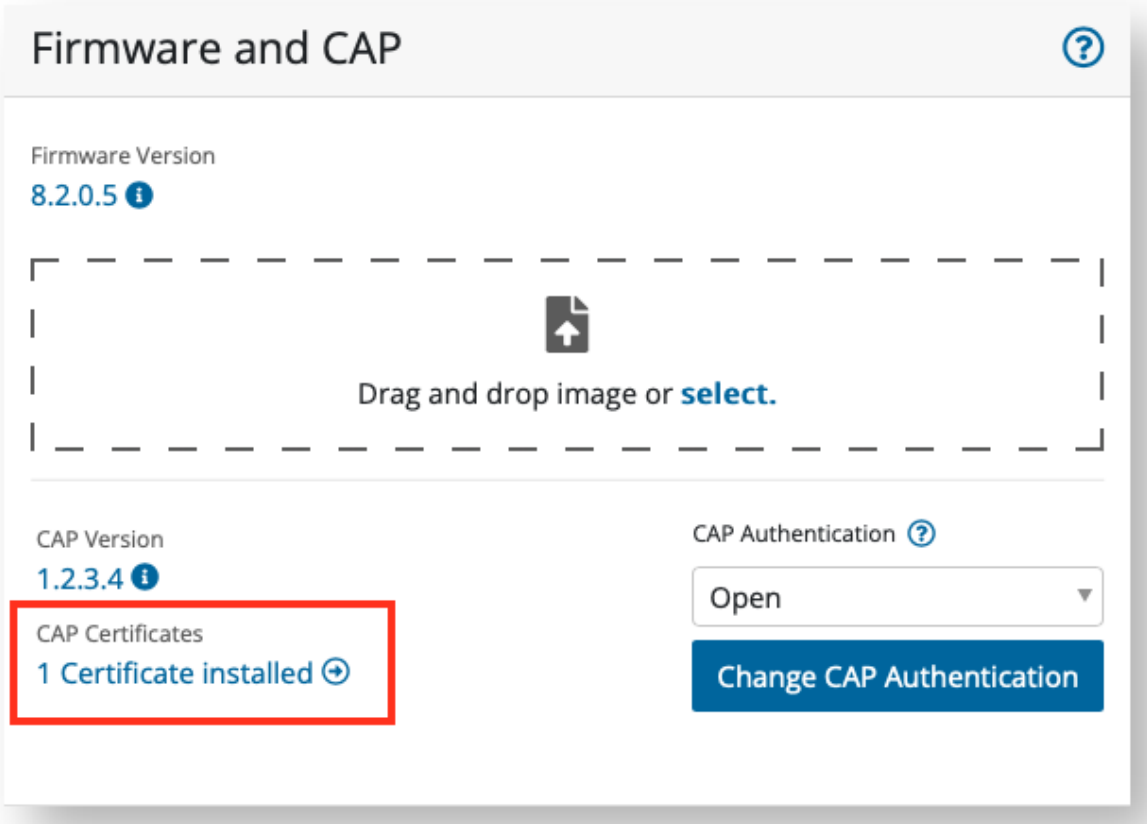
4.7.5 Installing a custom-signed CAP

In contrast to when you're installing a default-signed CAP, where `cap_gen` will use a default key to sign the upgrade file and the corresponding certificate is on the reader by default, you will need to upload your certificate onto the reader before the custom-signed CAP can be installed. After that, the method for installing the CAP image is the same as a firmware image.

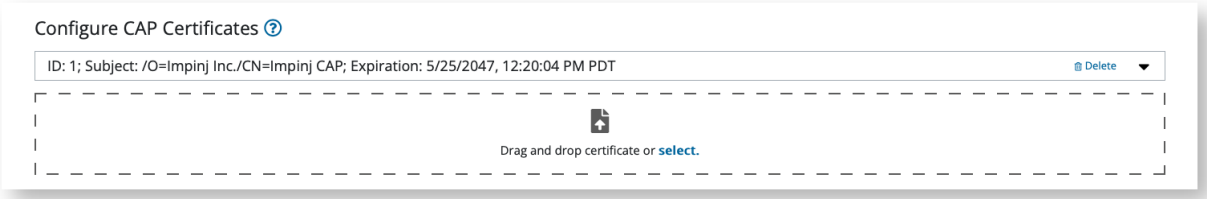
Note: The reader's CAP authentication mode will need to be either `Open` or `Secure`.

You can install a CAP by using either the WebUI or the IoT Device Interface (REST API).

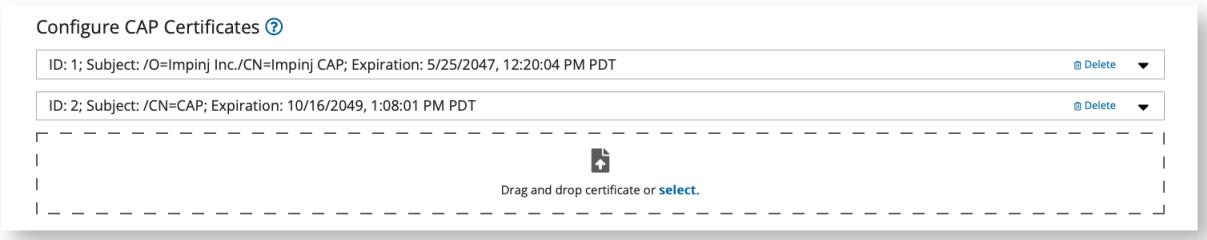
4.7.5.1 Installing the CAP authentication certificate via the WebUI Navigate to the reader's home webpage and you should see the section **Firmware and CAP**. Unless you have deleted the default certificate you should see that there's one certificate installed:

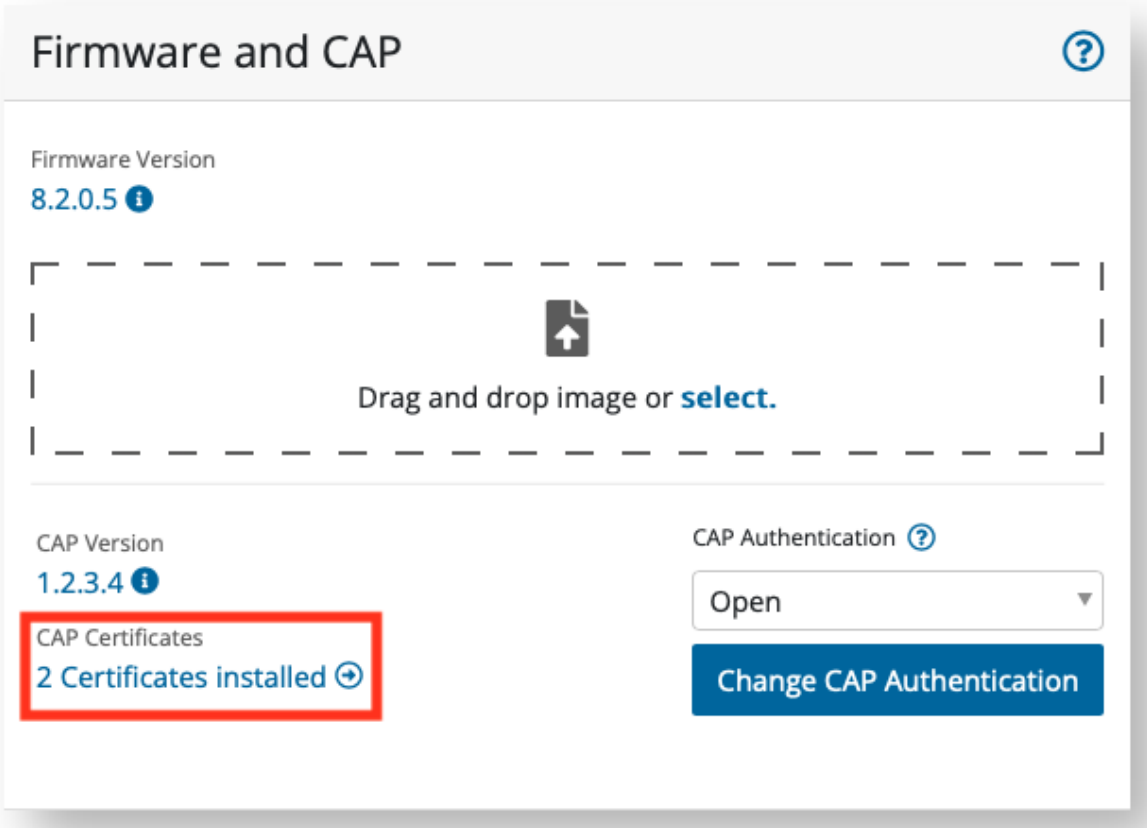


You can click on the the text that says "1 Certificate Installed →" to add, delete, and view the installed CAP certificates. It should have the default CAP certificate installed by default.



You can upload the one corresponding to the private key you used to sign your CAP image.





4.7.5.2 Installing a CAP authentication certificate with the IoT Device Interface A simple `curl` command will upload a CAP signing certificate to a reader.

```
$ curl -k -u : -X POST -F
"upload=@cap_signing/cap.cert.pem" https:///api/v1/system/certificates/cap/certs
```

4.8 CAP Encryption

For CAP encryption, the CAP authentication setting must be set to `Secure`. We're going to use the CAP example from the ETK to go through the CAP encryption process.

Generate the key pair to use for encrypting.

Note: This can be the same pair used for signing.

```
$ openssl req -x509 -newkey rsa:4096 -keyout cap.key.pem -out cap.cert.pem -sha256 -days 365
$ mkdir cap_encrypt
$ mv cap.cert.pem cap_encrypt
$ mv cap.key.pem cap_encrypt
```

In `<etk_dir>/examples/cap/cap_description.in`, uncomment the `Encrypt=True` line and uncomment and edit the `Recipients` line to point to the certificate created for encrypting. This line is basically telling the `cap_gen` script which certificate we want to use to encrypt the file.

```
# Encrypt the upgrade filesystem content. Note that this will be
# decrypted when the upgrade file is processed by the reader.
# The default is 'False' (don't encrypt)
Encrypt = True

# Should point to the cert from the key pair used to encrypt the bundle.
Recipients = ./cap_encrypt/cap.cert.pem
```

Now just generate the CAP as you would normally. You can either leave the bundle default-signed or sign it by passing the `-k` option illustrated in the previous section.

4.8.1 Generating the encrypted upgrade image

```
$ ../cap_gen.sh -d cap/cap_description.in -o my_cap.upgx
```

Because the bundle is now encrypted, the reader will need the certificate/private-key pair. The certificate is the one identified in `Recipients` and the private key is the one used to generate the certificate. You will need to bundle the two together in a `pkcs12` file.

Note: `openssl` will prompt you for a password with which to generate the bundle. The reader does not require one, but if you specify a password, it must be provided in the curl request when uploading.

```
$ openssl pkcs12 -export -out cap_encrypt/cap_enc_bundle.p12 \
-in cap_encrypt/cap.cert.pem -inkey cap_encrypt/cap.key.pem
```

You can currently only upload the bundle to the reader through the IoT Device Interface. It will return a `certId` that you will need to specify the certificate.

```
$ curl -k -u : -X POST \
-F "upload=@cap_encrypt/cap_enc_bundle.p12" \
-F"password=" \
https:///api/v1/system/certificates/tls/certs
```

You will need to tell the reader that you want to use the certificate you uploaded, with the `pkcs12` bundle, specifically for CAP encryption. The `certId` from our example is `1`.

```
$ curl -k -u : -X PUT -d "{"certId":}" \
https:///api/v1/system/certificates/tls/services/cap-encryption
```

Now you can upload the CAP to the reader as you would any other CAP or firmware image.

4.9 Accessing the Linux shell

Impinj readers have their own command-line interface named "RFID-Shell" or "RShell" for short. End users can access this interface with the RS-232 console port, or by connecting to the reader over the network via SSH if it is configured. For information on how to configure network services, see the **Software features** section.

With this interface, you can configure the reader and display system information. However, for an embedded developer, this interface is insufficient, so access to the underlying Linux shell is required. Because of the critical nature of this interface and the ability to adversely affect reader behavior if it is misused, access to the Linux shell (named "OSShell") is protected from the end user.

For an embedded developer, access to this interface is made possible via a custom application configuration file that is built into the CAP upgrade image. For details about creating the CAP partition, please see the section **CAP creation** above.

To enable access to OSShell via RShell, you must create the following file within the CAP file system: `/cust/sys/reader.conf`. This file does much more than enable access to OSShell. More specific information is provided in the **Reader configuration** section.

For OSShell access, the following lines (or *stanza*) must appear within the file.

```
[rshell]

password=developer
```

4.9.1 Enabling access to OSShell

After you build an upgrade image with the CAP partition containing this file, and load the upgrade image onto the reader, you can enable access to OSShell by using the following RShell command:

```
> osshell developer
```

The password (`developer` in this example) must match the password that exists in the reader configuration file loaded onto the reader. Invalid passwords will be rejected as if the command were invalid for security purposes. The OSShell feature is intended for embedded developers only. If the password is left blank (with the string `password=`), then password checking is disabled.

Note: Impinj highly recommends that the OSShell feature be disabled in deployed CAP image files.

4.9.1.1 Linux syslog An on-reader application can log information to the Linux application syslog using the BusyBox `logger` command line tool. The application syslog content is stored in the `/mnt/spp/log/syslog-app.log` file.

```
Usage: logger [OPTIONS] [MESSAGE]
  -s         Log to stderr as well as the system log
  -t TAG     Log using the specified tag (defaults to user name)
  -p PRIO    Priority (numeric or facility.level pair)
```

Example usage:

```
# logger -s "Test Output to SysLog 1"

# logger -s -t TEST1 "Test Output to SysLog 2"

# logger -s -t TEST1 -p 7 "Test Output to SysLog 3"

# logger -s -t TEST1 -p user.notice "Test Output to SysLog 04"
```

Resulting content in the syslog-app.log file:

```
Jul 18 15:59:18 (none) root: Test Output to SysLog 1
Jul 18 15:59:19 (none) TEST1: Test Output to SysLog 2
Jul 18 15:59:19 (none) TEST1: Test Output to SysLog 3
Jul 18 15:59:20 (none) TEST1: Test Output to SysLog 4
```

Note: The syslog priority levels are Emerg (0), Alert (1), Crit (2), Error (3), Warning (4), Notice (5), Info (6), and Debug (7). For more information on syslog levels, refer to the ***Impinj RShell Reference Manual***.

4.9.1.2 Secure File Transfer Protocol (SFTP)

There is an SFTP server on the reader that you can use to transfer files. As a security measure, this service is disabled by default. For information about how to enable SFTP, see the ***RShell Reference Manual***.

Note: As with the OSShell password, Impinj does not recommend deploying readers with this feature enabled.

4.9.2 Automatically starting a custom application

The custom application above is intended to be run manually, and is for demonstration purposes only. A typical custom application would be started along with the other runtime reader applications, as described in the **Runtime applications** section.

To configure a custom application to start when the reader is reset, the Linux boot process invokes the `/cust/start` application at system startup. This application is provided by the embedded developer and must have executable permissions. It is started only once in the background, and its responsibility is to launch the custom applications as required. This may itself be the custom application, or it could be a shell script that launches, and perhaps monitors, other applications.

The following code is an example of this type of script. This example tests for the presence of a custom application, and if it is executable, starts it. If the application ends, it logs a message and re-starts the application after a delay.

```
#!/bin/sh

export LD_LIBRARY_PATH=/cust/lib
(( count = 1 ))

while true ; do

   if [ -f /cust/app/rfid_control ] ; then

      if [ -x /cust/app/rfid_control ] ; then

         /cust/app/rfid_control

         /usr/bin/logger -p user.notice \
"Restarting custom application, count $count."

         (( count = count + 1 ))

      fi

      sleep 10

   else
      exit 0
   fi
done
```

Note: To make this sample work, change the logging level to a level that makes logged events with priority level `user.notice` visible. The description of how to configure the logging level threshold is in the section describing the `CONFIGURE LOGGING` command in the ***RShell Reference Manual***.

Example customer start application

On R700, the application can be monitored using systemd:

```
root@SpeedwayR-00-00-00:cust# systemctl status cap.service
```

4.9.3 Custom application library dependencies

The Impinj reader firmware contains a minimal set of libraries required for the applications it supports. A custom application might require additional libraries to operate that are not included in the SOP. Embedded applications that require additional library support can include these libraries in their CAP image, and can configure Linux via the `LD_LIBRARY_PATH` environment variable to scan the CAP partition when it searches for dynamic library dependencies.

A typical example of this is a custom application written in C++ that requires the dynamic library `libstdc++`, which is not included in the SOP. To handle this scenario, developers should include this library in their CAP file system. Then the custom application start script should set the `LD_LIBRARY_PATH` environment variable to point to the directory where the shared objects are located, as is shown in the sample start script above.

4.9.4 Custom Application Execution Environment

CAP applications execute in a sandboxed environment with a limited view of the system. We use systemd to provide these facilities. Note that these restrictions are only present when the CAP application is executed by systemd as part of the `cap.service` file -- these restrictions will not be present if the CAP application is executed by any other mechanism (such as manually from the shell).

At a high level, the sandbox serves two purposes:

1. restricting access to resources (including CPU and RAM)
2. isolating the CAP from the rest of the system

4.9.4.1 CAP Resource Restrictions The CAP application is allotted the equivalent of 50% of one processor core and 256MB of RAM.

4.9.4.2 CAP Isolation The CAP has a read-only view of the filesystem except for some special locations. These include:

1. `/dev/log` to enable the CAP to write to the system log
2. `/dev/eeprom` to enable the CAP to run RShell commands (RShell must be able to access this file with write permissions set)
3. `/var/run` to allow access to certain Unix Domain sockets
4. `/mnt/spp/conf` and `/var/conf` to allow the CAP to change system configurations. For example, some RShell commands will update these directories
5. `/mnt/spp/log` to allow access to the system log
6. `/mnt/spp/core` to enable writing core files for debugging in case the CAP crashes
7. `/scratch` to store large files or files that need to persist (note that `/cust` can and should be used for this). The filesystem mounted at `/scratch` is 864MB and the filesystem mounted at `/cust` is up to 256MB
8. `/etc` to allow the CAP to change general system configurations

The CAP has a private `/tmp` directory that is isolated from the global `/tmp` directory used by the system. This directory is created when the CAP starts and is removed when CAP execution terminates. When the CAP is running, it can be accessed from OSShell and is located at `/tmp/systemd-private-<hash>-cap.service-<hash>`. This is a tmpfs and should be used instead of `/dev/shm`, which is not guaranteed to be present.

4.9.4.3 Complete Systemd Execution Environment Configuration A complete listing of the systemd Service configurations can be found on your reader at `/lib/systemd/system/cap.service`. For more information about these options, see the systemd documentation.