CAPRunner Documentation

Release 0.1

Benoit Allard

March 02, 2015

Contents

Contents:

CHAPTER 1

Introduction

1.1 Motivation

This project was born when a need appeared for a JavaCard Applet simulator that would let us have a look at the internal bits such as maximum stack depth, memory used, ... A first idea was to parse the Java source file and interpret it. This idea quickly disapeared in flavor of CAP file parsing and execution. This way, we take the exact same imput as the smartcard for a result as comparable as possible. The CAP file parser was born.

1.2 Overview

The main component of this package is **runcap.py**, a little script (less than 300 lines) that agregate the included libs into a CAP file executer. This script has dependencies on pythoncard (https://bitbucket.org/benallard/pythoncard) which provide the basic OS functionnalities of the JavaCard.

1.3 Integration

Thie project takes part of the WebSCard (https://bitbucket.org/benallard/webscard) project. This particular part cares that SmartCard can be emulated without the need of a physical one.

1.4 Document structure

This document is split into two parts: The *User Manual* (page 5) and the *Development Manual* (page 9).

As a user you a probably only interested in the first one, as a developper, both one should be of interest. The *Development Manual* (page 9) will go more into details on how to build your application using part of the CAPRunner.

CHAPTER 2

User Manual

2.1 Introduction

This part of this document describes how to *use* CAPRunner from a User point of vue.

2.2 Installation

From the README:

2.2.1 Installation

Your best chance for the moment is to clone the repository using Mercurial (http://mercurial.selenic.com/) to a local directory:

```
$ hg clone https://bitbucket.org/benallard/caprunner/
```

This will create a copy of the repository in a `caprunner` directory.

In order to get the dependencies right, I suggest you also clone their repositories (the second one is optionnal):

```
$ hg clone https://bitbucket.org/benallard/pythoncard/
$ hg clone https://bitbucket.org/benallard/pythonplatform/
```

Finally, the easiest way to get the dependencies resolved is to *copy* (or *link* on UNIX) the following directories at the root of the `caprunner` directory:

- `python`, `pythoncard` and `pythoncardx` from the pythoncard directory
- and (optionnaly) `org` from the pythonplatform directory.

2.2.2 Getting started

As a final step, you will need to *compile* the export files used during compilation of your applet into a JSON file. This step is needed in order to not read every single export file at each startup.

This is done with the following command line:

```
$ genref.py --dump dump.json /path/to/export_files
```

This will generate a file called `dump.json` containing the necessary information from the export files. The path given as second parameter is the directory containing the export files from your JavaCard Development Kit (e.g. the `api21_export_files` directory for a JavaCard 2.1.2).

CAPRunner will expect the generated json file to be called `<JavaCard version>.json`

Finally, you should be able to launch `runcap.py`. By default, it will emulate a JavaCard Classic 3.0.1, if you need another version, just give it as parameter. For instance:

```
$ ./runcap.py 2.1.2
```

Starting from here, `runcap` will wait for your orders. Please refer to the *Command format* (page 6) for the format of the order to send to `runcap`.

2.3 RunCAP

2.3.1 Overview

2.3.2 Parameters

2.3.3 Command format

APDUs

APDU are sent to the CAP file in the following format:

```
==> se nd ap du;
<== ex pe ct ed re tu rn co de;
```

Each byte is in hexadecimal format without `0x` preceding them.

If the CAP file returns another return code, the execution will stop.

Load command

To load an applet into the interpreter, the following command has to be used:

```
load: /path/to/capfile.cap
```

The result of this command will be that the given CAP file will be loaded into memory and its applets will be ready for the *Install command* (page 6). At this step, the whole CAP file will be parsed.

Install command

To install an applet into the interpreter, and make it selectable, the following command has to be used:

```
install: data : offset
```

The `data` is the one tramsmitted to `Applet.install()` [1] prepended with `AIDlen || AID` of the applet to install. The `offset` parameter indicate the offset of the `AIDlen` byte inside the data.

Print the log

There is also a command to print an internal log of what happen during the last APDU processing:

```
log;
```

Example

The following example is used to test the HelloWorld example from the JavaCard Development Kit.

```
1 load: HelloWorld-2.1.2.cap
2
3 install: 0a a0 00 00 00 62 03 01 0c 01 01 00 00 00 : 00
4
5 ==> 00 a4 04 00 0a a0 00 00 00 62 03 01 0c 01 01 7F
6 <== 00 A4 04 00 0A A0 00 00 00 62 03 01 0C 01 01 90 00
7
8 ==> 80 B8 00 00 0C 0A
9 <== 80 B8 00 00 0C 90 00
```

- Line 1, the CAP file `HelloWorld-2.1.2.cap` is loaded inside the interpreter.
- Line 3, the applet with AID `a0 00 00 00 62 03 01 0c 01 01` is being installed. This applet is included inside the CAP file. The data transmitted to the function is constructed as follow:
  - First, the length of the AID on one byte: `0a`
  - Then comes the AID itself `a0 00 00 00 62 03 01 0c 01 01`
  - And finally the data to be transmitted to the `Applet.install()` function.

  As second parameter comes the offset of the AIDlength bytes, `00` in our case as we did'nt gave junk data in the input.
- Line 5-6, the first APDU is being set to the Applet, namely a `SELECT` command with the AID of the installed applet. this select is processed by the interpreter and then by the Applet itself, returning some data.
- Line 8-9, a random command is processed by the Applet and some data is returned.

2.4 RefCollection

2.5 Export

[1] This is the concatenation of the following three fields, each one prepended with its own length (which is `0` if the field is empty)

- instance AID (if not specified the applet AID will be taken)
- control info
- applet data

CHAPTER 3

Development Manual

CHAPTER 4

Indices and tables

- *genindex*
- *modindex*
- *search*