User Guide

Alif Security Toolkit

Table of Contents

1. Introduction

The Alif Security Toolkit (SETOOLS) contains several tools (written in Python3) required to program and provision an Alif Semiconductor SoC device.

Provisioning means being able to add images to the internal NVM storage (MRAM) as well as adding Security assets such as Keys to the One Time Programmable memory (OTP).

ALIF images, known as System TOC (STOC), for the device are already provisioned into the MRAM, a user cannot change these, but can update to a new STOC version supplied by Alif. User images such as Application programs (binaries written for A32 or M55 cores) need to be packaged and written to the MRAM. The packages are called an Application Table of Contents (ATOC).

There are also pre-built binary images for the ALIF Table of Contents (STOC) Package. The STOC contains the latest binary version of SERAM. The STOC also contains debug stubs which will execute on the Application cores to allow connection from the Debugger. These stubs are only loaded to Application CPUs that are not configured to run.

NOTE: These tools are designed to work with the following environment:

- Windows PC
- LINUX
- CPUBoard with Alif Ensemble or Crescendo REV_A1 device populated

- Python3 must be installed.
- Python cryptography library must be installed (See section 3 below)

Please ensure you have followed the installation instructions in the Alif Security Toolkit Quick Start Guide before proceeding.

This release version is app***-release-SE_FW_0.42.000_DEV.zip***

2. Quick User Guide

Please ensure you have followed the Installation Guide (Section 3) before proceeding.

2.1 Updating to the latest ALIF release

```
$ cd <release-location>/app-release
$ python3 updateSystemPackage.py
```

This will upgrade the System (ALIF) images in MRAM to the latest version in this release. It will not touch your Application MRAM images. This uses the In System Programming (ISP) scheme using the SE-UART.

2.2 Adding / Changing an application image

To change or update an image to add to MRAM do the following:

```
$ cd app-release
$ edit build/config/app-cfg.json and add new binaries images

$ python3 app-gen-toc.py

(Or use below command to use a different configuration)

$ python3 app-gen-toc.py -f build/config/app-myfile.json

$ python3 app-write-mram.py
```

The app-gen-toc tool performs the packaging of your binary ready for writing to the MRAM.

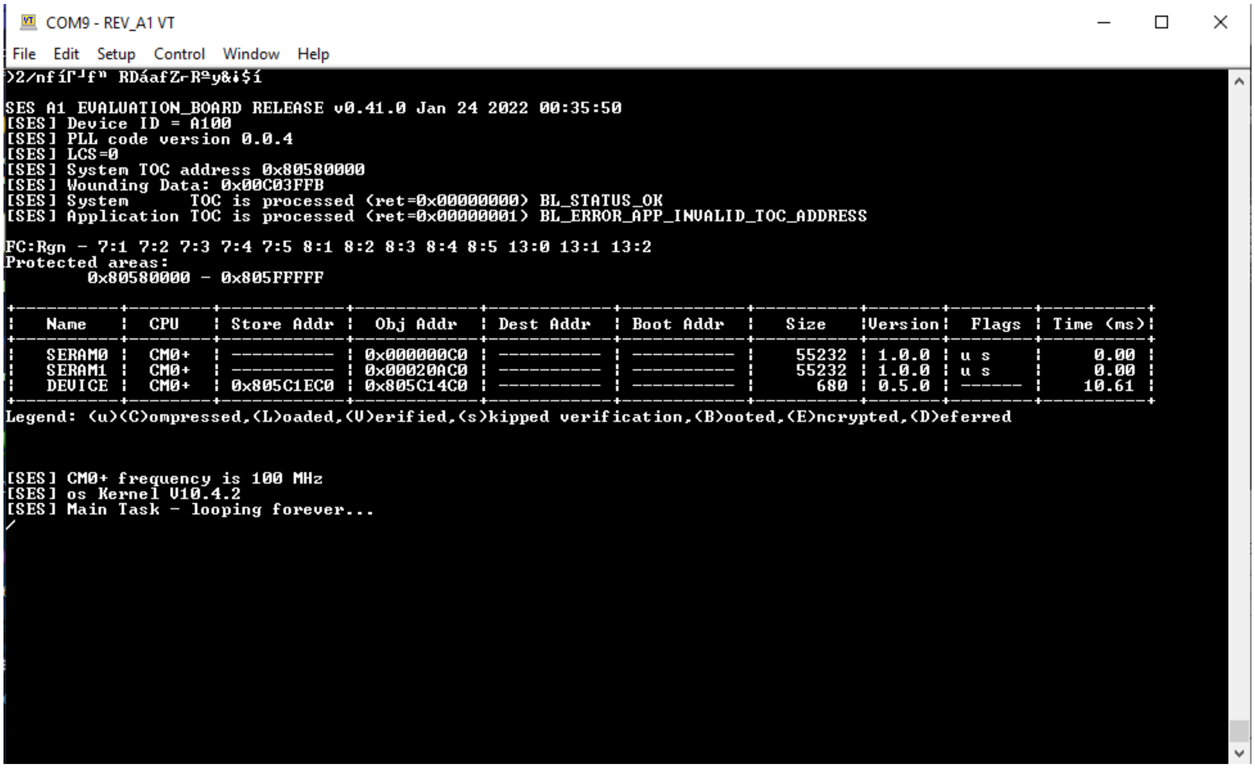
The app-write-mram.py tool performs the writing of the application package to the MRAM. After this is completed, it will reset your target platform and the contents of the MRAM will be executed.

3. TOC Packages

The device will arrive already provisioned. This means that the SES is already programmed into the MRAM.

3.1 System Table of Contents (STOC)

On initial power up you will see the following SE-UART output:



```
>2/nfíΓ┘f" RDáafZ┌Rºy&¡$í

SES A1 EVALUATION_BOARD RELEASE v0.41.0 Jan 24 2022 00:35:50
[SES] Device ID = A100
[SES] PLL code version 0.0.4
[SES] LCS=0
[SES] System TOC address 0x80580000
[SES] Wounding Data: 0x00C03FFB
[SES] System      TOC is processed (ret=0x00000000) BL_STATUS_OK
[SES] Application TOC is processed (ret=0x00000001) BL_ERROR_APP_INVALID_TOC_ADDRESS

FC:Rgn - 7:1 7:2 7:3 7:4 7:5 8:1 8:2 8:3 8:4 8:5 13:0 13:1 13:2
Protected areas:
        0x80580000 - 0x805FFFFF

+----------+--------+------------+------------+------------+------------+----------+-------+--------+----------+
¦   Name   ¦  CPU   ¦ Store Addr ¦  Obj Addr  ¦ Dest Addr  ¦ Boot Addr  ¦   Size   ¦Version¦ Flags  ¦ Time (ms)¦
+----------+--------+------------+------------+------------+------------+----------+-------+--------+----------+
¦   SERAM0 ¦  CM0+  ¦ ---------- ¦ 0x000000C0 ¦ ---------- ¦ ---------- ¦    55232 ¦ 1.0.0 ¦ u s    ¦     0.00 ¦
¦   SERAM1 ¦  CM0+  ¦ ---------- ¦ 0x00020AC0 ¦ ---------- ¦ ---------- ¦    55232 ¦ 1.0.0 ¦ u s    ¦     0.00 ¦
¦   DEVICE ¦  CM0+  ¦ 0x805C1EC0 ¦ 0x805C14C0 ¦ ---------- ¦ ---------- ¦      680 ¦ 0.5.0 ¦ ------ ¦    10.61 ¦
+----------+--------+------------+------------+------------+------------+----------+-------+--------+----------+
Legend: (u)(C)ompressed,(L)oaded,(V)erified,(s)kipped verification,(B)ooted,(E)ncrypted,(D)eferred



[SES] CM0+ frequency is 100 MHz
[SES] os Kernel V10.4.2
[SES] Main Task - looping forever...
/
```

The Error code reported from the Application TOC processing is indicating that there is no user supplied ATOC images in MRAM.

Version 38: In this case, SES will automatically launch debug stubs into the available Application cores. This means you can attach to these cores with the ARM-DS debugger. If there are ATOC components to boot, SES will launch these as instructed by the attributes of the image. Once this is completed any Application cores that are not running will get a Debug stub.

Version 40: The debug stubs are now removed from the STOC. The user must generate the debug stubs and add then to the ATOC.

3.2 SE-UART Output Fields

The output from the SE-UART shows what SES has processed during the boot sequence of the device.

Banner

Shows the version of the SES software.

PLL

Shows the version of the PLL.

LCS (Life Cycle State)

Shows the LCS state of the device.

[SES] Output

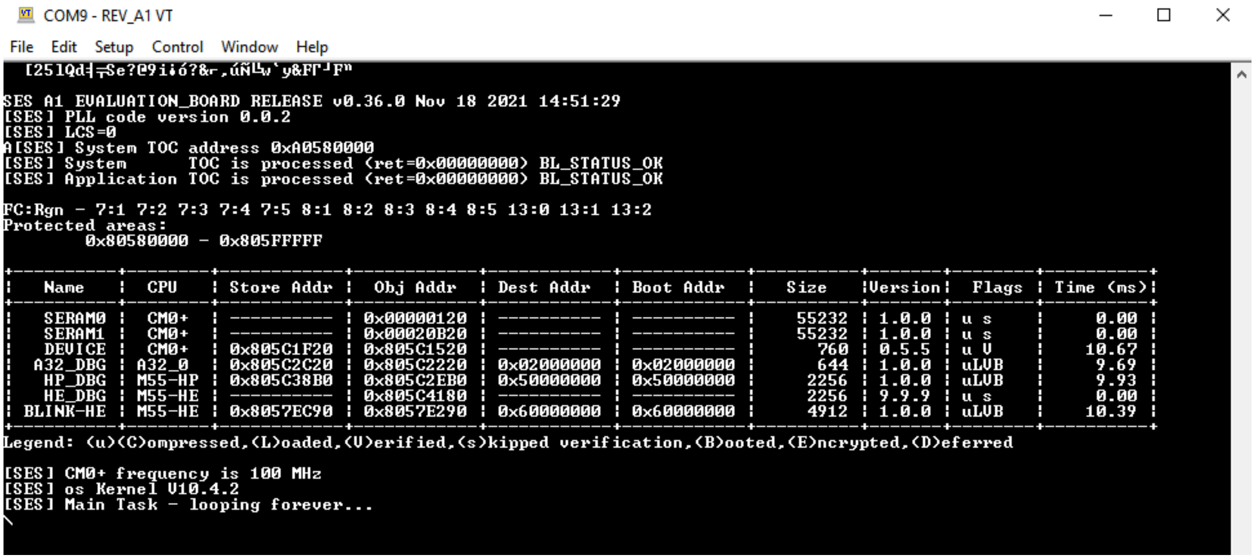
Shows the state of processing the STOC and ATOC images found in MRAM.

[SES] Table

Shows the details of all TOC objects processed from MRAM

Name	ASCII string name from the JSON file.
CPU	Which CPU is being targeted.
Store Address	Where in MRAM is this object stored
Obj Address	TOC address
Dest Addr	Where in RAM is this object being copied to.
Boot Addr	What address is being used to Boot from.
Size	Size of the binary object
Version	Version ID specified in the JSON file
Flags	
C	Image is Compressed
u	Image is Uncompressed
L	Image is LOADED to a RAM location
V	Image is VERIFIED
s	Image is SKIPPED
B	Image has been BOOTED
D	Image is DEFERRED, Booting will happen later
Time	Time in microseconds from the start of TOC processing until completion, this could include booting.

If you do add an ATOC image what you will see is the following:



```
  [251Qd╡╤Se?@9i↓ó?&r,úÑ╚w`y&FΓ┘F"

SES A1 EVALUATION_BOARD RELEASE v0.36.0 Nov 18 2021 14:51:29
[SES] PLL code version 0.0.2
[SES] LCS=0
A[SES] System TOC address 0xA0580000
[SES] System      TOC is processed (ret=0x00000000) BL_STATUS_OK
[SES] Application TOC is processed (ret=0x00000000) BL_STATUS_OK

FC:Rgn - 7:1 7:2 7:3 7:4 7:5 8:1 8:2 8:3 8:4 8:5 13:0 13:1 13:2
Protected areas:
        0x80580000 - 0x805FFFFF

+----------+--------+------------+------------+------------+------------+------------+-------+----------+----------+
¦   Name   ¦  CPU   ¦ Store Addr ¦  Obj Addr  ¦ Dest Addr  ¦ Boot Addr  ¦    Size    ¦Version¦  Flags   ¦ Time (ms)¦
+----------+--------+------------+------------+------------+------------+------------+-------+----------+----------+
¦   SERAM0 ¦  CM0+  ¦ ---------- ¦ 0x00000120 ¦ ---------- ¦ ---------- ¦      55232 ¦ 1.0.0 ¦ u s      ¦     0.00 ¦
¦   SERAM1 ¦  CM0+  ¦ ---------- ¦ 0x00020B20 ¦ ---------- ¦ ---------- ¦      55232 ¦ 1.0.0 ¦ u s      ¦     0.00 ¦
¦   DEVICE ¦  CM0+  ¦ 0x805C1F20 ¦ 0x805C1520 ¦ ---------- ¦ ---------- ¦        760 ¦ 0.5.5 ¦ u V      ¦    10.67 ¦
¦  A32_DBG ¦ A32_0  ¦ 0x805C2C20 ¦ 0x805C2220 ¦ 0x02000000 ¦ 0x02000000 ¦        644 ¦ 1.0.0 ¦ uLVB     ¦     9.69 ¦
¦   HP_DBG ¦ M55-HP ¦ 0x805C38B0 ¦ 0x805C2EB0 ¦ 0x50000000 ¦ 0x50000000 ¦       2256 ¦ 1.0.0 ¦ uLVB     ¦     9.93 ¦
¦   HE_DBG ¦ M55-HE ¦ ---------- ¦ 0x805C4180 ¦ ---------- ¦ ---------- ¦       2256 ¦ 9.9.9 ¦ u s      ¦     0.00 ¦
¦ BLINK-HE ¦ M55-HE ¦ 0x8057EC90 ¦ 0x8057E290 ¦ 0x60000000 ¦ 0x60000000 ¦       4912 ¦ 1.0.0 ¦ uLVB     ¦    10.39 ¦
+----------+--------+------------+------------+------------+------------+------------+-------+----------+----------+
Legend: (u)(C)ompressed,(L)oaded,(V)erified,(s)kipped verification,(B)ooted,(E)ncrypted,(D)eferred

[SES] CM0+ frequency is 100 MHz
[SES] os Kernel V10.4.2
[SES] Main Task - looping forever...
\
```

In this example, we have added an M55_HE “Blinky” program (see BLINK-HE).

The debug stub for the M55_HE is no longer loaded as there is a real ATOC image to execute for this application cpu. The HE_DBG entry though is still reported by SERAM, but it’s Attributes (Flags) show that it was not (**B**)ooted or (**L**)oaded. This is because the Debug stubs are stored in the STOC package and SERAM processes all TOC information in MRAM

3.3 Application Table of Contents (ATOC)

The Application TOC package contains a Factory image.

For BETA releases this consists of:

- HE_55 LED Blinky program.

This code is stored in MRAM and copied to a RAM location (The TCM for the HE_55) and booted.

To rebuild this ATOC package

```
$ cd app-release
$ python3 app-gen-toc.py -f build/config/app-cfg.json
$ python3 app-write-image.py
```

3.4 ATOC Placed Images

ATOC packages allow for the absolute placement of user images in MRAM. See “mramAddress” option in the app-gen-toc.py configuration file.

Users must ensure that these images conform to the MRAM alignment rules and always be 16-byte aligned.

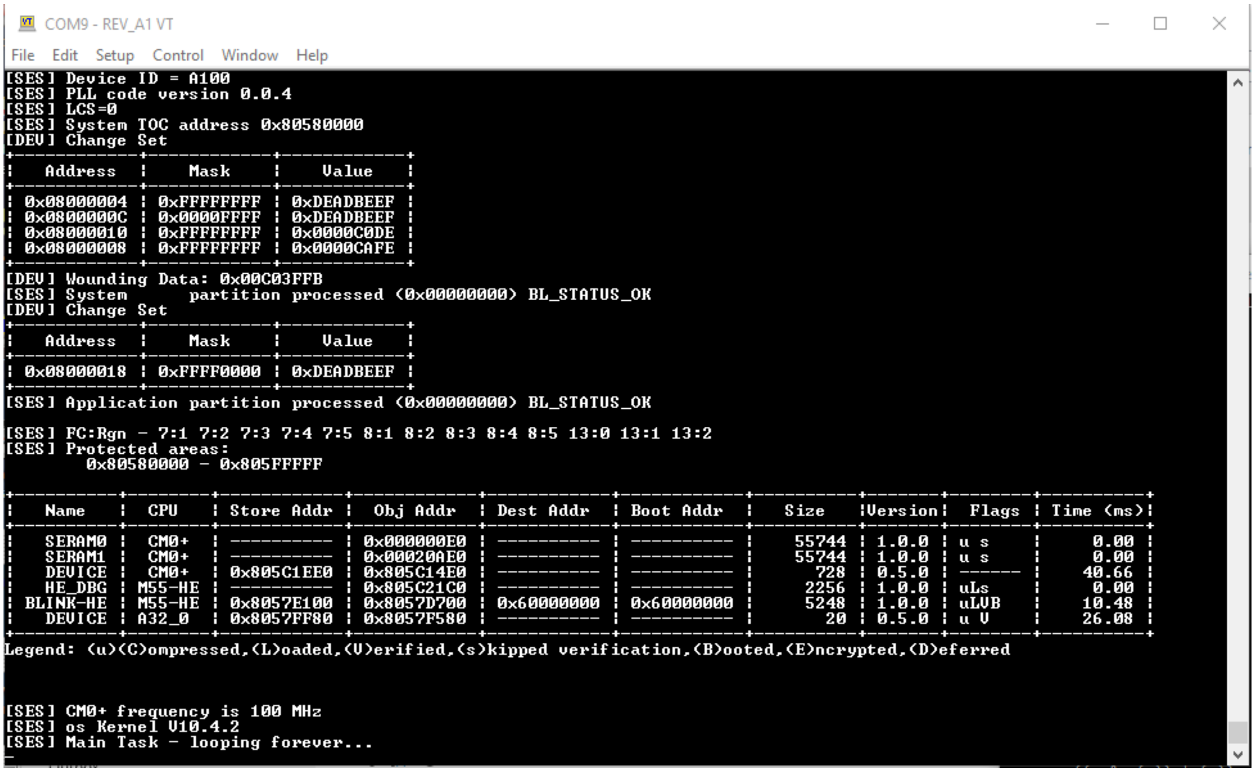
Images that are managed by the ATOC tool i.e., not placed at an absolute address are guaranteed to be aligned correctly.

3.5 Change Sets

Change Sets are used to allow for Static initialization of registers at Boot up time.

This is known as Static device configuration. Change Set data is processed and executed as part of the TOC processing. A User could use this for peripheral / register / memory initialization without the necessity to write any code. Pin muxing, Firewall, Clock, Event and Interrupt masking can be programmed using this method.

Change Sets can be added to the STOC packages (by ALIF only) and to the Application TOC packages by Users.



```
[SES] Device ID = A100
[SES] PLL code version 0.0.4
[SES] LCS=0
[SES] System TOC address 0x80580000
[DEV] Change Set
+------------+------------+------------+
¦  Address   ¦    Mask    ¦   Value    ¦
+------------+------------+------------+
¦ 0x08000004 ¦ 0xFFFFFFFF ¦ 0xDEADBEEF ¦
¦ 0x0800000C ¦ 0x0000FFFF ¦ 0xDEADBEEF ¦
¦ 0x08000010 ¦ 0xFFFFFFFF ¦ 0x0000C0DE ¦
¦ 0x08000008 ¦ 0xFFFFFFFF ¦ 0x0000CAFE ¦
+------------+------------+------------+
[DEV] Wounding Data: 0x00C03FFB
[SES] System      partition processed (0x00000000) BL_STATUS_OK
[DEV] Change Set
+------------+------------+------------+
¦  Address   ¦    Mask    ¦   Value    ¦
+------------+------------+------------+
¦ 0x08000018 ¦ 0xFFFF0000 ¦ 0xDEADBEEF ¦
+------------+------------+------------+
[SES] Application partition processed (0x00000000) BL_STATUS_OK

[SES] FC:Rgn - 7:1 7:2 7:3 7:4 7:5 8:1 8:2 8:3 8:4 8:5 13:0 13:1 13:2
[SES] Protected areas:
        0x80580000 - 0x805FFFFF

+----------+--------+------------+------------+------------+------------+----------+-------+--------+-----------+
¦   Name   ¦  CPU   ¦ Store Addr ¦  Obj Addr  ¦ Dest Addr  ¦ Boot Addr  ¦   Size   ¦Version¦ Flags  ¦ Time (ms)¦
+----------+--------+------------+------------+------------+------------+----------+-------+--------+-----------+
¦   SERAM0 ¦  CM0+  ¦ ---------- ¦ 0x000000E0 ¦ ---------- ¦ ---------- ¦    55744 ¦ 1.0.0 ¦ u s    ¦      0.00 ¦
¦   SERAM1 ¦  CM0+  ¦ ---------- ¦ 0x00020AE0 ¦ ---------- ¦ ---------- ¦    55744 ¦ 1.0.0 ¦ u s    ¦      0.00 ¦
¦   DEVICE ¦  CM0+  ¦ 0x805C1EE0 ¦ 0x805C14E0 ¦ ---------- ¦ ---------- ¦      728 ¦ 0.5.0 ¦ ------ ¦     40.66 ¦
¦   HE_DBG ¦ M55-HE ¦ ---------- ¦ 0x805C21C0 ¦ ---------- ¦ ---------- ¦     2256 ¦ 1.0.0 ¦ uLs    ¦      0.00 ¦
¦ BLINK-HE ¦ M55-HE ¦ 0x8057E100 ¦ 0x8057D700 ¦ 0x60000000 ¦ 0x60000000 ¦     5248 ¦ 1.0.0 ¦ uLVB   ¦     10.48 ¦
¦   DEVICE ¦ A32_0  ¦ 0x8057FF80 ¦ 0x8057F580 ¦ ---------- ¦ ---------- ¦       20 ¦ 0.5.0 ¦ u V    ¦     26.08 ¦
+----------+--------+------------+------------+------------+------------+----------+-------+--------+-----------+
Legend: (u)(C)ompressed,(L)oaded,(V)erified,(s)kipped verification,(B)ooted,(E)ncrypted,(D)eferred


[SES] CM0+ frequency is 100 MHz
[SES] os Kernel V10.4.2
[SES] Main Task - looping forever...
_
```

What is a Change Set?

A Change Set is a group of 32-bit values representing an Address, a Mask, and a Value. With these three elements, any memory or register address can be modified using Change Sets.

Example – Setting the Ethernet external Clock

The Ethernet external clock can be enabled using the following Change Set configuration;

```
"ChangeSets" : [
     {
        "address" : "0x71007408",
        "mask" : "0x00000001",
        "value" : "0x00000001"
     }
]
```

After SERAM processes the above Change Set, the Ethernet external clock will be enabled.

Adding a Change Set

If we take the application configuration file app-cfg.json

```
{
        "BLINK-HE": {
                "binary": "m55_blink_he.bin",
                "version" : "1.0.0",
                "cpu_id": "M55_HE",
                "loadAddress": "0x60000000",
                "flags": ["load", "boot"]
        }
},
        "DEVICE": {
                "binary": "device-config.json",
                "version" : "0.5.00"
},
```

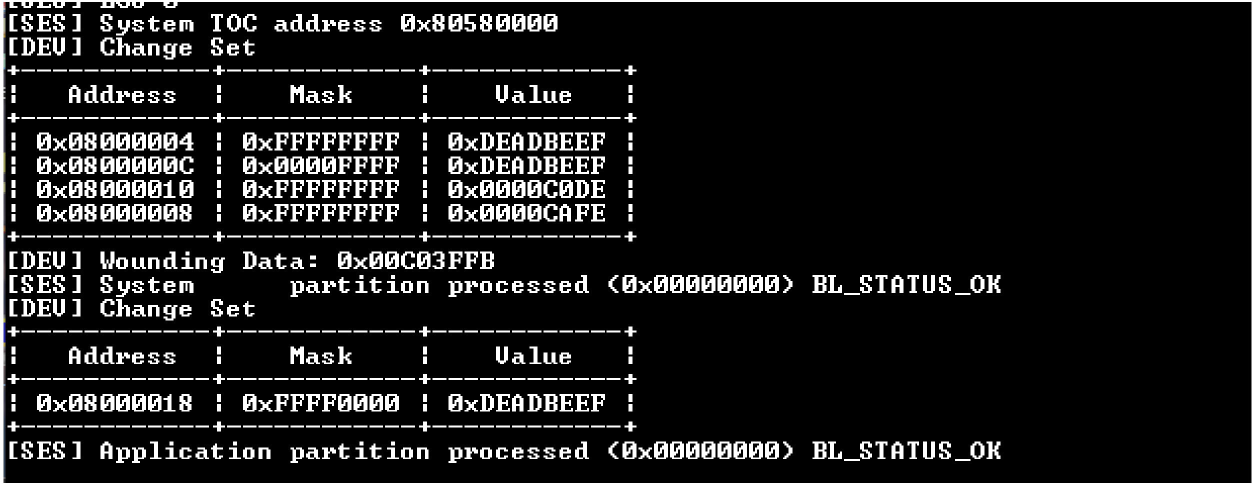
Here we have added a DEVICE configuration section.

The device-config.json file is as follows:

```
{
    "clocks": {
        "ChangeSets" : [
        {
            "address" : "0x08000018",
            "mask" : "0xFFFFFFFF",
            "value" : "0xdeadbeef"
        }
        ]
}
```

For this example, we are adding a Change Set to a location in Global memory SRAM_1 (0x080000018).

When SES boots it will process the DEVICE Configuration section in MRAM



```
[SES] System TOC address 0x80580000
[DEV] Change Set
+-------------+-------------+-------------+
|   Address   |    Mask     |    Value    |
+-------------+-------------+-------------+
| 0x08000004  | 0xFFFFFFFF  | 0xDEADBEEF  |
| 0x0800000C  | 0x0000FFFF  | 0xDEADBEEF  |
| 0x08000010  | 0xFFFFFFFF  | 0x0000C0DE  |
| 0x08000008  | 0xFFFFFFFF  | 0x0000CAFE  |
+-------------+-------------+-------------+
[DEV] Wounding Data: 0x00C03FFB
[SES] System      partition processed (0x00000000) BL_STATUS_OK
[DEV] Change Set
+-------------+-------------+-------------+
|   Address   |    Mask     |    Value    |
+-------------+-------------+-------------+
| 0x08000018  | 0xFFFF0000  | 0xDEADBEEF  |
+-------------+-------------+-------------+
[SES] Application partition processed (0x00000000) BL_STATUS_OK
```

In this example, there are some DEVICE configurations for the ALIF System TOC then the Application TOC is processed we see the Change Set being applied.

4. ISP (In System Programming)

ISP is a mechanism for connecting to the Secure Enclave using the Secure Enclave UART (SE-UART) and performing various operations with the device.

Please ensure the prerequisite python packages are installed before starting. See the installation instructions in the Alif Security Toolkit Quick Start Guide.

SE-UART Connection

UART Settings (REV_A1 Device)

Baud Rate	115200 (default)
Data	8 bits
Parity	None
Stop Bits	1 bit
Flow control	None

NOTE: SEROM uses a Baud Rate of 100000

UART Settings (REV_A0 Device)

Baud Rate	50000 (default)
Data	8 bits
Parity	None
Stop Bits	1 bit
Flow control	None

Dynamic Baud Rate Change

Scripts updating MRAM will increase the baud rate from the default value to 921600 baud. Once the operation is completed the Baud rate is set back to the default value.

Use the ‘-s’ option to override this dynamic change facility.

Baud Rate Override

The ISP method uses a default baud rate. For REV_A devices this is 50000.

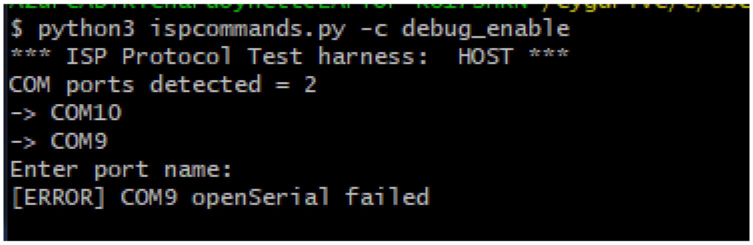
You can override the default baud rate by doing either:

- Edit the local configuration file isp_config_data.cfg and change baudrate.
- Use the -b option e.g. -b 115200

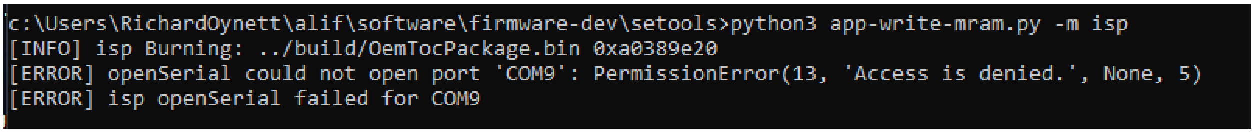
All ISP tools have this option.

UART Errors

There is only one SE-UART on the device. If you are using ISP, please ensure you have no other Tera term or putty sessions using the same SE-UART. The following shows the output if the SE-UART is already being used by another program:



```
$ python3 ispcommands.py -c debug_enable
*** ISP Protocol Test harness:  HOST ***
COM ports detected = 2
-> COM10
-> COM9
Enter port name:
[ERROR] COM9 openSerial failed
```



```
c:\Users\RichardOynett\alif\software\firmware-dev\setools>python3 app-write-mram.py -m isp
[INFO] isp Burning: ../build/OemTocPackage.bin 0xa0389e20
[ERROR] openSerial could not open port 'COM9': PermissionError(13, 'Access is denied.', None, 5)
[ERROR] isp openSerial failed for COM9
```

This indicates that the UART is already being used (e.g., a Tera-Term session is still running).

ISP Discovery

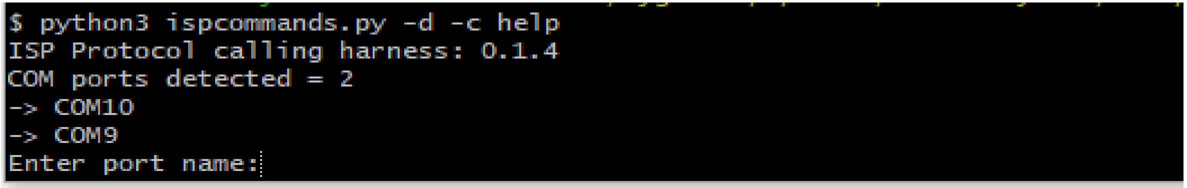
This command discovers the available Serial communication ports. The first time you execute an SETOOLS script you will be prompted for the required serial port.

When the ports are presented, just enter the port name and press [ENTER].

This port data is saved in a local configuration file (isp_config_data.cfg).

The next time an SETOOLS command is invoked if this configuration file is present, it will use the parameters from this file.

To override this option simply use the -d option:



```
$ python3 ispcommands.py -d -c help
ISP Protocol calling harness: 0.1.4
COM ports detected = 2
-> COM10
-> COM9
Enter port name:
```

This will force a re-discovery of the Serial ports.

All the MRAM ISP tools have the discovery option which will prompt you for the serial port.

5. Tool Flow

The tools allow a user to provision the device.

This includes programming MRAM with application binaries for A32 and M55 embedded processors available for user applications.

a. Use tools-config.py to review and modify current tool options.

b. Generate RoT with gen-rot.py only once (or anytime new keys are required) – **Not required** as this release packet already contains a sample RoT and corresponding keys and certificates, but it can be used if user wants to re-generate the RoT.

c. Edit `build/config/app-cfg.json` file (or a json file with other name) with images to include in the desired **AppTocPackage.bin** image and copy the binaries into the build/ directory. Run app-toc.py to generate the package.

d. Power on the board and connect the Serial interface for the following steps.

d.1 (Optional) Use updateSystemPackage.py to update latest Alif Secure Enclave firmware.

d.2 Use app-write-mram.py to upload the AppTocPackage.bin image into the MRAM.

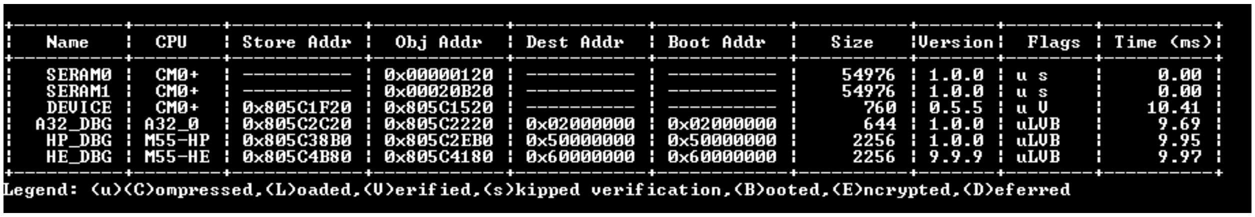
d.2.1 (Optional) Reset the board to reboot the chip to verify that the SERAM image loads and boots properly.

5.1 Debug Stubs

To enable connection with the debugger such as ARM-DS, the target core needs to be running.

This is achieved by running debug stubs on the cores. These are part of System Table of Contents, if the Application TOC is not populated then the debug stubs for the relevant Application CPU are executed.

The following is the SE-UART output on boot when the Application MRAM is not programmed



```
+-----------+---------+------------+------------+------------+------------+-----------+--------+---------+-----------+
|   Name    |  CPU    | Store Addr |  Obj Addr  | Dest Addr  | Boot Addr  |   Size    |Version|  Flags  | Time (ms)|
+-----------+---------+------------+------------+------------+------------+-----------+--------+---------+-----------+
|   SERAM0  |  CM0+   | ---------- | 0x00000120 | ---------- | ---------- |     54976 | 1.0.0 | u s     |      0.00 |
|   SERAM1  |  CM0+   | ---------- | 0x00020B20 | ---------- | ---------- |     54976 | 1.0.0 | u s     |      0.00 |
|   DEVICE  |  CM0+   | 0x805C1F20 | 0x805C1520 | ---------- | ---------- |       760 | 0.5.5 | u V     |     10.41 |
|  A32_DBG  | A32_0   | 0x805C2C20 | 0x805C2220 | 0x02000000 | 0x02000000 |       644 | 1.0.0 | uLVB    |      9.69 |
|   HP_DBG  | M55-HP  | 0x805C38B0 | 0x805C2EB0 | 0x50000000 | 0x50000000 |      2256 | 1.0.0 | uLVB    |      9.95 |
|   HE_DBG  | M55-HE  | 0x805C4B80 | 0x805C4180 | 0x60000000 | 0x60000000 |      2256 | 9.9.9 | uLVB    |      9.97 |
+-----------+---------+------------+------------+------------+------------+-----------+--------+---------+-----------+
Legend: (u)(C)ompressed,(L)oaded,(V)erified,(s)kipped verification,(B)ooted,(E)ncrypted,(D)eferred
```

The debug stubs, denoted by the _DBG names, are loaded by SESfor each relevant Application core.

5.2 MRAM Burners

Two tools used to write MRAM

- updateSystemPackage.py
- app-write-mram.py

One will update the System TOC (STOC) and the other will update the Users Application MRAM area.

When using a generated TOC package this will be guaranteed to correctly aligned and padded for writing to MRAM.

In the case of User supplied images the burning tool itself will take care of any alignment issues such as sending extra bytes to align/pad the image already sent.

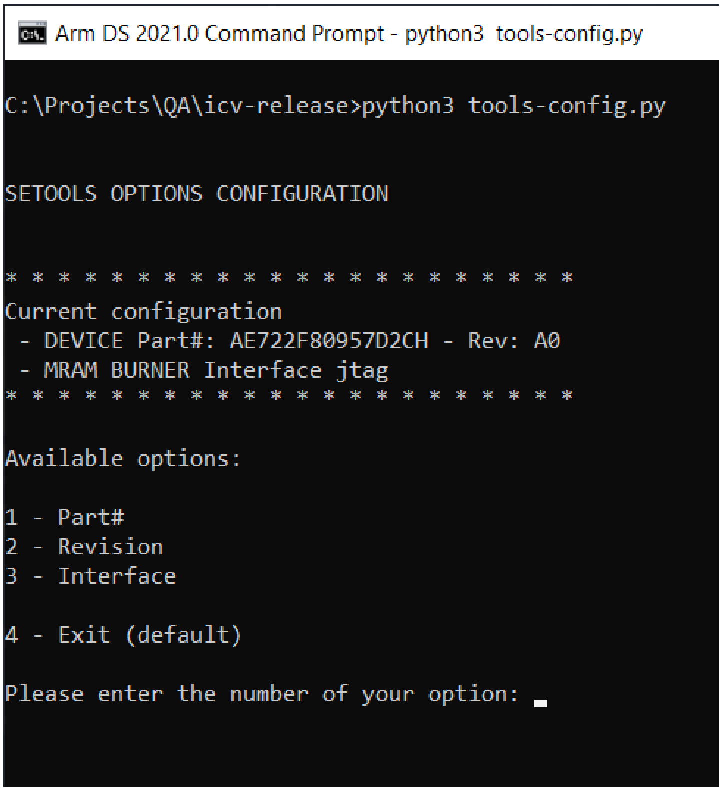
6. Running the Tools

tools-config.py

From the command prompt.

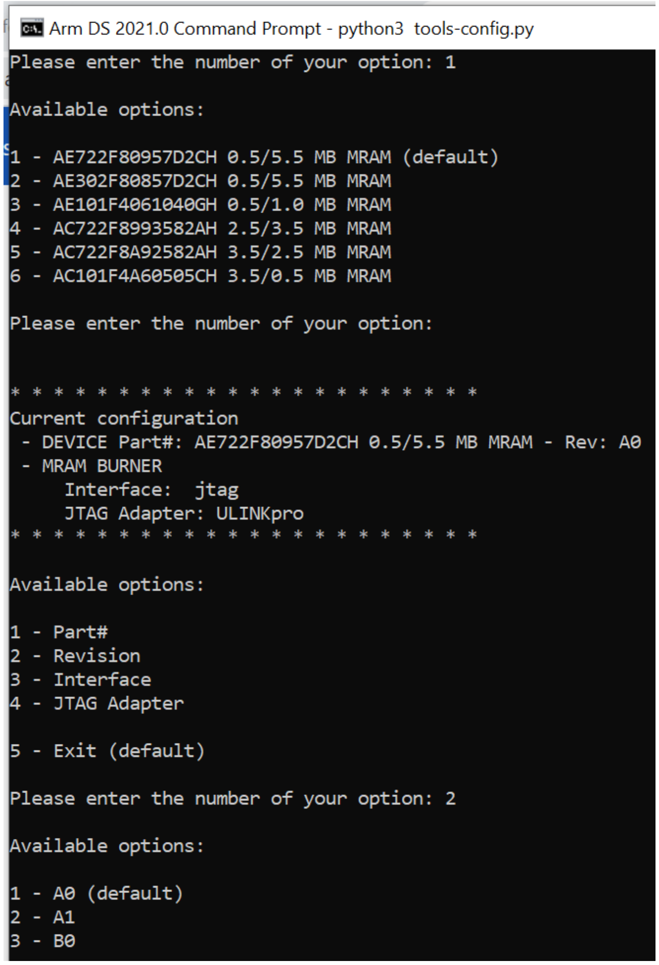
```
$ python3 tools-config.py
```

This script will show the current tool options and allow for their modification.



```
C:\Projects\QA\icv-release>python3 tools-config.py


SETOOLS OPTIONS CONFIGURATION


* * * * * * * * * * * * * * * * * * * * * *
Current configuration
 - DEVICE Part#: AE722F80957D2CH - Rev: A0
 - MRAM BURNER Interface jtag
* * * * * * * * * * * * * * * * * * * * * *

Available options:

1 - Part#
2 - Revision
3 - Interface

4 - Exit (default)

Please enter the number of your option:
```

Press the menu number to show available options.



```
Please enter the number of your option: 1

Available options:

1 - AE722F80957D2CH 0.5/5.5 MB MRAM (default)
2 - AE302F80857D2CH 0.5/5.5 MB MRAM
3 - AE101F4061040GH 0.5/1.0 MB MRAM
4 - AC722F8993582AH 2.5/3.5 MB MRAM
5 - AC722F8A92582AH 3.5/2.5 MB MRAM
6 - AC101F4A60505CH 3.5/0.5 MB MRAM

Please enter the number of your option:


* * * * * * * * * * * * * * * * * * * * * * *
Current configuration
 - DEVICE Part#: AE722F80957D2CH 0.5/5.5 MB MRAM - Rev: A0
 - MRAM BURNER
     Interface:  jtag
     JTAG Adapter: ULINKpro
* * * * * * * * * * * * * * * * * * * * * * *

Available options:

1 - Part#
2 - Revision
3 - Interface
4 - JTAG Adapter

5 - Exit (default)

Please enter the number of your option: 2

Available options:

1 - A0 (default)
2 - A1
3 - B0
```

Choose the desired option pressing the option number (or press Enter key to choose the default one, shown in parentheses).

Once the tool options are configured, the rest of the tools will take these options automatically.

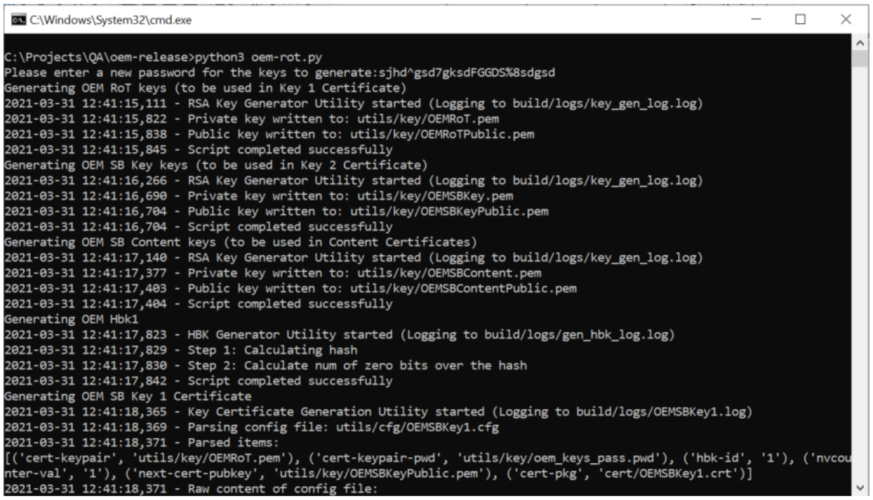
(Note: every time a new Part Number or device Revision is changed, it’s recommended to re-generate the APP TOC Package using the app-toc.py tool).

app-gen-rot.py

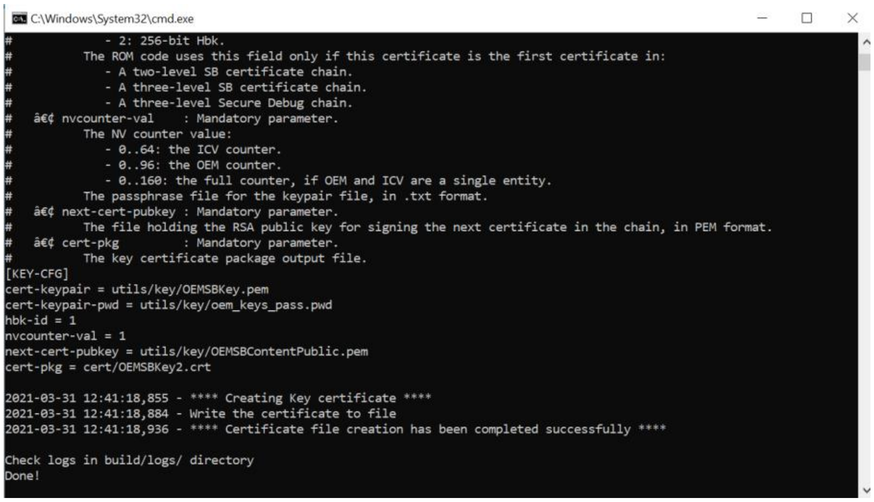
From the command prompt;

```
$ python3 app-gen-rot.py  (-h for help)
```

For the first time, it will ask for a password for the generated keys and it will be saved for later use. If a new password is desired, delete the *utils/key/oem_keys_pass.pwd* file and run this script again.



```
C:\Projects\QA\oem-release>python3 oem-rot.py
Please enter a new password for the keys to generate:sjhd^gsd7gksdFGGDS%8sdgsd
Generating OEM RoT keys (to be used in Key 1 Certificate)
2021-03-31 12:41:15,111 - RSA Key Generator Utility started (Logging to build/logs/key_gen_log.log)
2021-03-31 12:41:15,822 - Private key written to: utils/key/OEMRoT.pem
2021-03-31 12:41:15,838 - Public key written to: utils/key/OEMRoTPublic.pem
2021-03-31 12:41:15,845 - Script completed successfully
Generating OEM SB Key keys (to be used in Key 2 Certificate)
2021-03-31 12:41:16,266 - RSA Key Generator Utility started (Logging to build/logs/key_gen_log.log)
2021-03-31 12:41:16,690 - Private key written to: utils/key/OEMSBKey.pem
2021-03-31 12:41:16,704 - Public key written to: utils/key/OEMSBKeyPublic.pem
2021-03-31 12:41:16,704 - Script completed successfully
Generating OEM SB Content keys (to be used in Content Certificates)
2021-03-31 12:41:17,140 - RSA Key Generator Utility started (Logging to build/logs/key_gen_log.log)
2021-03-31 12:41:17,377 - Private key written to: utils/key/OEMSBContent.pem
2021-03-31 12:41:17,403 - Public key written to: utils/key/OEMSBContentPublic.pem
2021-03-31 12:41:17,404 - Script completed successfully
Generating OEM Hbk1
2021-03-31 12:41:17,823 - HBK Generator Utility started (Logging to build/logs/gen_hbk_log.log)
2021-03-31 12:41:17,829 - Step 1: Calculating hash
2021-03-31 12:41:17,830 - Step 2: Calculate num of zero bits over the hash
2021-03-31 12:41:17,842 - Script completed successfully
Generating OEM SB Key 1 Certificate
2021-03-31 12:41:18,365 - Key Certificate Generation Utility started (Logging to build/logs/OEMSBKey1.log)
2021-03-31 12:41:18,369 - Parsing config file: utils/cfg/OEMSBKey1.cfg
2021-03-31 12:41:18,371 - Parsed items:
[('cert-keypair', 'utils/key/OEMRoT.pem'), ('cert-keypair-pwd', 'utils/key/oem_keys_pass.pwd'), ('hbk-id', '1'), ('nvcou
nter-val', '1'), ('next-cert-pubkey', 'utils/key/OEMSBKeyPublic.pem'), ('cert-pkg', 'cert/OEMSBKey1.crt')]
2021-03-31 12:41:18,371 - Raw content of config file:
```



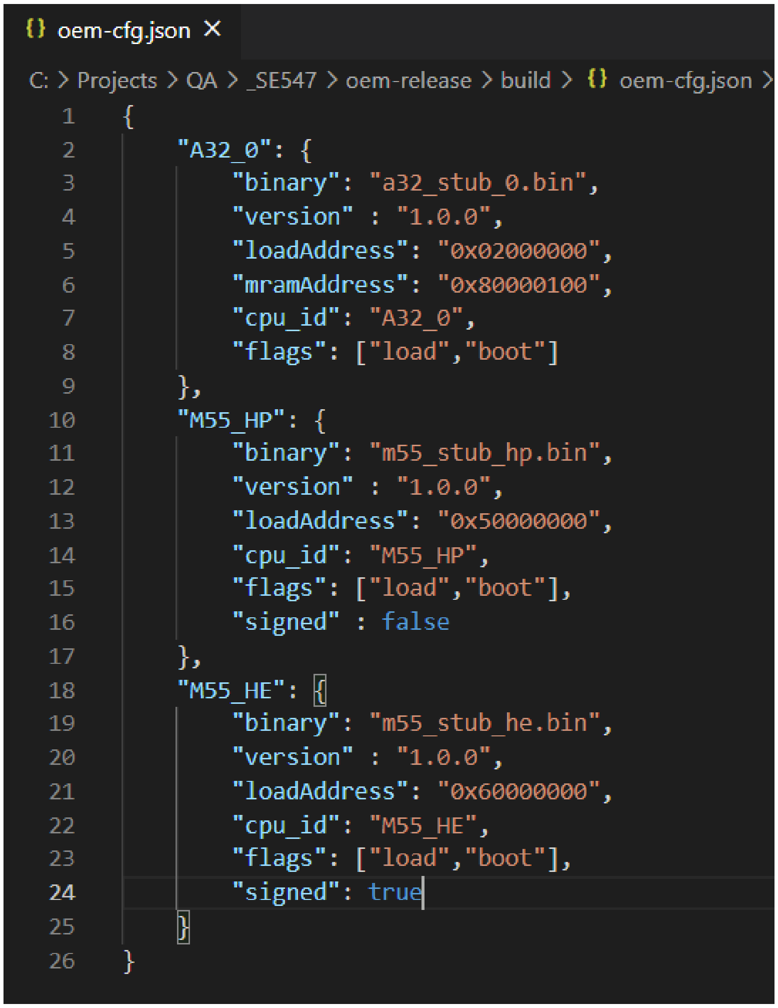
```
#              - 2: 256-bit Hbk.
#           The ROM code uses this field only if this certificate is the first certificate in:
#              - A two-level SB certificate chain.
#              - A three-level SB certificate chain.
#              - A three-level Secure Debug chain.
#    â€¢ nvcounter-val    : Mandatory parameter.
#           The NV counter value:
#              - 0..64: the ICV counter.
#              - 0..96: the OEM counter.
#              - 0..160: the full counter, if OEM and ICV are a single entity.
#           The passphrase file for the keypair file, in .txt format.
#    â€¢ next-cert-pubkey : Mandatory parameter.
#           The file holding the RSA public key for signing the next certificate in the chain, in PEM format.
#    â€¢ cert-pkg         : Mandatory parameter.
#           The key certificate package output file.
[KEY-CFG]
cert-keypair = utils/key/OEMSBKey.pem
cert-keypair-pwd = utils/key/oem_keys_pass.pwd
hbk-id = 1
nvcounter-val = 1
next-cert-pubkey = utils/key/OEMSBContentPublic.pem
cert-pkg = cert/OEMSBKey2.crt

2021-03-31 12:41:18,855 - **** Creating Key certificate ****
2021-03-31 12:41:18,884 - Write the certificate to file
2021-03-31 12:41:18,936 - **** Certificate file creation has been completed successfully ****

Check logs in build/logs/ directory
Done!
```

Once finished, keys will be generated in *utils/key/*, certificates are generated in *cert/*, and logs in *build/logs/* folder.

app-gen-toc.py

To generate the APP TOC Package, the `build/config/app-cfg.json` file must be configured.

Edit this file:


{} oem-cfg.json ×
C: > Projects > QA > _SE547 > oem-release > build > {} oem-cfg.json >

```
{
    "A32_0": {
        "binary": "a32_stub_0.bin",
        "version" : "1.0.0",
        "loadAddress": "0x02000000",
        "mramAddress": "0x80000100",
        "cpu_id": "A32_0",
        "flags": ["load","boot"]
    },
    "M55_HP": {
        "binary": "m55_stub_hp.bin",
        "version" : "1.0.0",
        "loadAddress": "0x50000000",
        "cpu_id": "M55_HP",
        "flags": ["load","boot"],
        "signed" : false
    },
    "M55_HE": {
        "binary": "m55_stub_he.bin",
        "version" : "1.0.0",
        "loadAddress": "0x60000000",
        "cpu_id": "M55_HE",
        "flags": ["load","boot"],
        "signed": true
    }
}
```

This configuration file is in JSON format and must specify each of the binary images the user wants to include in the APP TOC Package. SERAM will process the images in the specified order (from top to bottom).

A JSON file may contain multiple objects, where each object is represented by a set of name-value pairs.

In the configuration file, each image must be specified as an object in the following way;

```
“IMAGE_IDENTIFIER”: {
      “attribute_1”: “value_1”,
      “attribute_2”: “value_2”,
      ……………
      “attribute_n”: “value_n”
}
```

Multiple images will be declared by multiple objects, separated by comma. For example;

```
{
   “IMAGE_1”: {

   },
   “IMAGE_2”: {

   },
   ………………
   “IMAGE_N”: {

   }
}
```

The “IMAGE_IDENTIFIER” is an 8-character length (MAX) description intended to give a short description to the image. The tool will truncate or extend this field’s length as appropriate. This name ***should be unique*** per configuration file.

Inside each image object, different attributes can be used to define the parameters of the binary image. The following attributes are valid:

binary: [MANDATORY] - this attribute declares the name of the binary to be included in the APP TOC Package. It is assumed the binary exists in the build/images/ folder.

version: [OPTIONAL] – this attribute declares the version of the image, and it should follow the ‘X.Y.Z’ format. Example; ‘1.0.0’

loadAddress: [MANDATORY if LOAD flag is set] – this attribute specifies the loading address in (RAM) memory where the code will be executed from. Global Memory Addresses (in hexadecimal format) should be used for this attribute.

mramAddress: [OPTIONAL] – this attribute specifies the MRAM address location where the user wishes the image to reside, and addresses (in hexadecimal format) starting from 0x8000-0100 (for REV_Ax) can be used. This option is to support the “Fixed Address Image” or “absolute placement” use case, where the user needs to position the binary in a specific address in MRAM, so other running code can locate this image in the specified location (Linux use case). If this attribute is used with the BOOT flag, XIP (execution in place) is supported. If this flag is not set, then the image will reside at this address, with no further implication. If this attribute is NOT specified, the tool will define the MRAM position to allocate the binary in memory.

flags: [OPTIONAL] – the following flags can be used – example [“load”, “boot”]:

- LOAD – The image will be loaded into the specified memory address
- BOOT – The specified CPU_ID will be started to execute the image
- ENCRYPT – The image will be encrypted
- COMPRESS – The image will be compressed
- DEFERRED – The image will be skipped at boot time (i.e., no boot or load) and wait for a service request at runtime

cpu_id: [OPTIONAL] – this attribute indicates the CPU to start once the binary is ready to be executed. Valid values are:

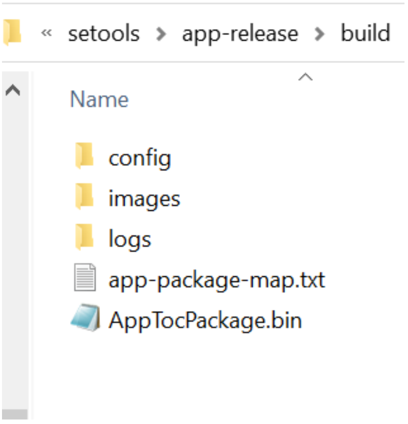
[A32_0, A32_1, A32_2, A32_3, M55_HP, M55_HE]

Note: Only one CPU_ID should be specified.

signed: [OPTIONAL] – this attribute is a binary value (true/false) specifying if the image is signed (“signed”: true) or unsigned (“signed”: false). In a typical case for Secure Boot, all images should be signed, so there is no need to specify this attribute. Only if the user wishes to specify an unsigned image, this attribute should be set as false.

disabled: [OPTIONAL] – this attribute allows a temporary disable of an image in the configuration file for testing purposes, so there is no need to delete the object for that image. Specifying “disabled”: true, the tool will ignore the entry. Either deleting the attribute, or setting it as false, the tool will include back the image in the final APP Package.

The JSON configuration file, along with all the binaries declared in it, should be allocated in the build/ folder (build/config/ and build/images respectively);



Once the *app-cfg.json* file is configured as desired, we need to be sure that all declared images do exist In the *build/images/* folder.

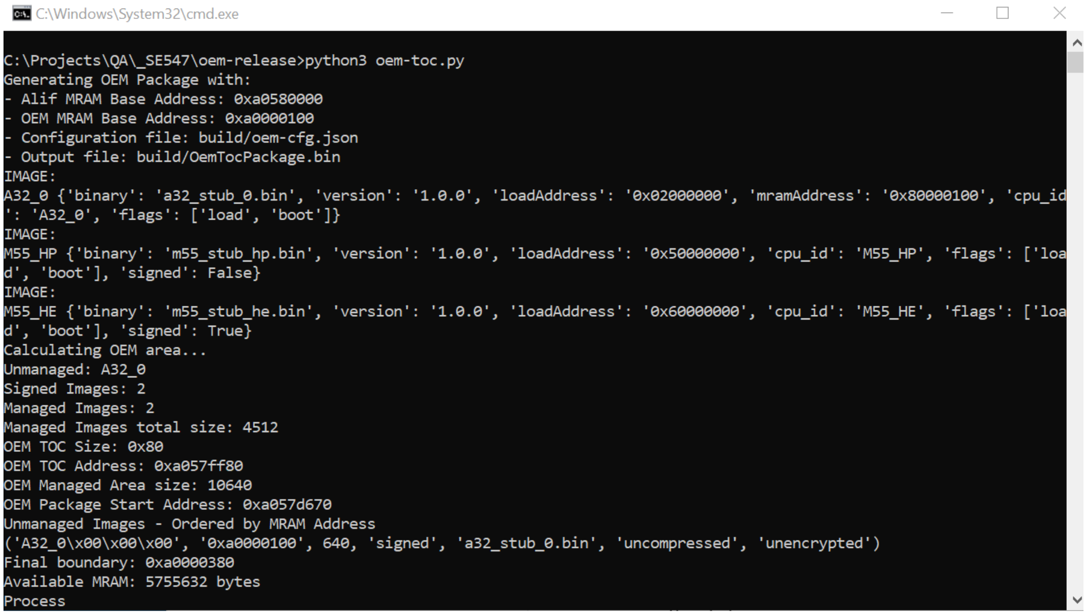
After running the *app-gen-toc.py* tool, the **APP TOC Package** (AppTocPackage.bin) will be generated in the *build/* folder.

The *build/logs/* folder contains a log for the Generation (OEMSBContent.log file)

Once the configuration is done, and all declared images exist in *build/images/* folder, continue with the APP TOC Package generation step;

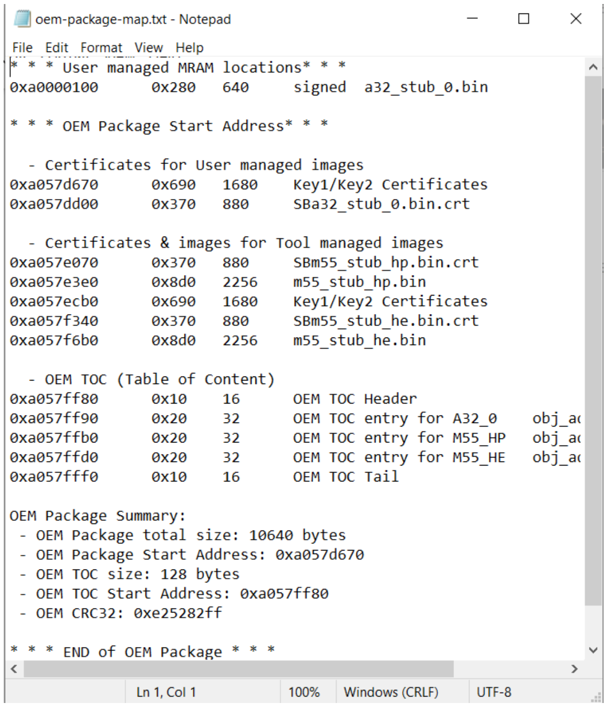
From the command prompt;

```
$ python3 app-gen-toc.py  (-h for help)
```



```
C:\Projects\QA\_SE547\oem-release>python3 oem-toc.py
Generating OEM Package with:
- Alif MRAM Base Address: 0xa0580000
- OEM MRAM Base Address: 0xa0000100
- Configuration file: build/oem-cfg.json
- Output file: build/OemTocPackage.bin
IMAGE:
A32_0 {'binary': 'a32_stub_0.bin', 'version': '1.0.0', 'loadAddress': '0x02000000', 'mramAddress': '0x80000100', 'cpu_id
': 'A32_0', 'flags': ['load', 'boot']}
IMAGE:
M55_HP {'binary': 'm55_stub_hp.bin', 'version': '1.0.0', 'loadAddress': '0x50000000', 'cpu_id': 'M55_HP', 'flags': ['loa
d', 'boot'], 'signed': False}
IMAGE:
M55_HE {'binary': 'm55_stub_he.bin', 'version': '1.0.0', 'loadAddress': '0x60000000', 'cpu_id': 'M55_HE', 'flags': ['loa
d', 'boot'], 'signed': True}
Calculating OEM area...
Unmanaged: A32_0
Signed Images: 2
Managed Images: 2
Managed Images total size: 4512
OEM TOC Size: 0x80
OEM TOC Address: 0xa057ff80
OEM Managed Area size: 10640
OEM Package Start Address: 0xa057d670
Unmanaged Images - Ordered by MRAM Address
('A32_0\x00\x00\x00', '0xa0000100', 640, 'signed', 'a32_stub_0.bin', 'uncompressed', 'unencrypted')
Final boundary: 0xa0000380
Available MRAM: 5755632 bytes
Process
```

Once the tool finishes, a map file named *app-package-map.txt* will be created in build/ folder to provide a reference of the memory map and images/certificates packaged in the APP TOC Package created by the tool. This memory map file will have the following structure;



```
* * * User managed MRAM locations* * *
0xa0000100      0x280   640     signed  a32_stub_0.bin

* * * OEM Package Start Address* * *

  - Certificates for User managed images
0xa057d670      0x690   1680    Key1/Key2 Certificates
0xa057dd00      0x370   880     SBa32_stub_0.bin.crt

  - Certificates & images for Tool managed images
0xa057e070      0x370   880     SBm55_stub_hp.bin.crt
0xa057e3e0      0x8d0   2256    m55_stub_hp.bin
0xa057ecb0      0x690   1680    Key1/Key2 Certificates
0xa057f340      0x370   880     SBm55_stub_he.bin.crt
0xa057f6b0      0x8d0   2256    m55_stub_he.bin

  - OEM TOC (Table of Content)
0xa057ff80      0x10    16      OEM TOC Header
0xa057ff90      0x20    32      OEM TOC entry for A32_0    obj_a
0xa057ffb0      0x20    32      OEM TOC entry for M55_HP   obj_a
0xa057ffd0      0x20    32      OEM TOC entry for M55_HE   obj_a
0xa057fff0      0x10    16      OEM TOC Tail

OEM Package Summary:
 - OEM Package total size: 10640 bytes
 - OEM Package Start Address: 0xa057d670
 - OEM TOC size: 128 bytes
 - OEM TOC Start Address: 0xa057ff80
 - OEM CRC32: 0xe25282ff

* * * END of OEM Package * * *
```

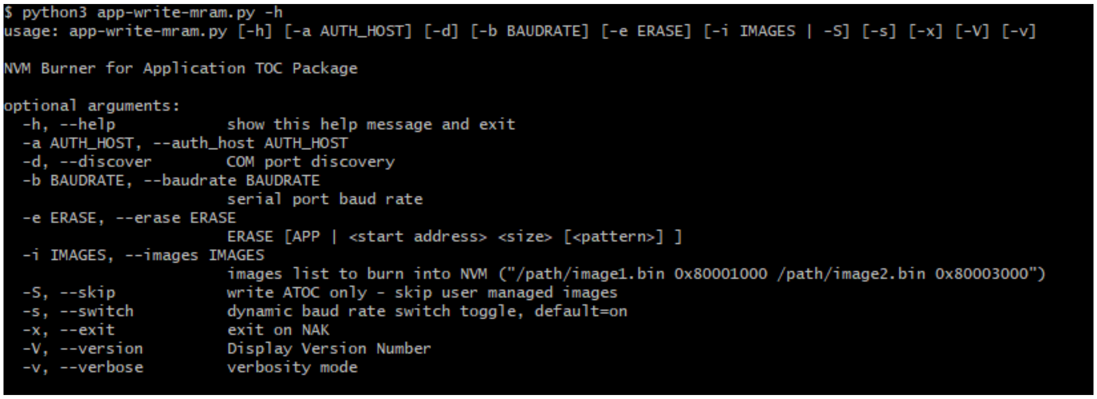
Additionally, a script file will be created in bin/ folder to automate the burn of this package into MRAM, using the app-write-mram.py tool. See next section for details.

app-write-mram.py

- This utility uses ISP (In System Programming) via the SE-UART connection, to update the MRAM.

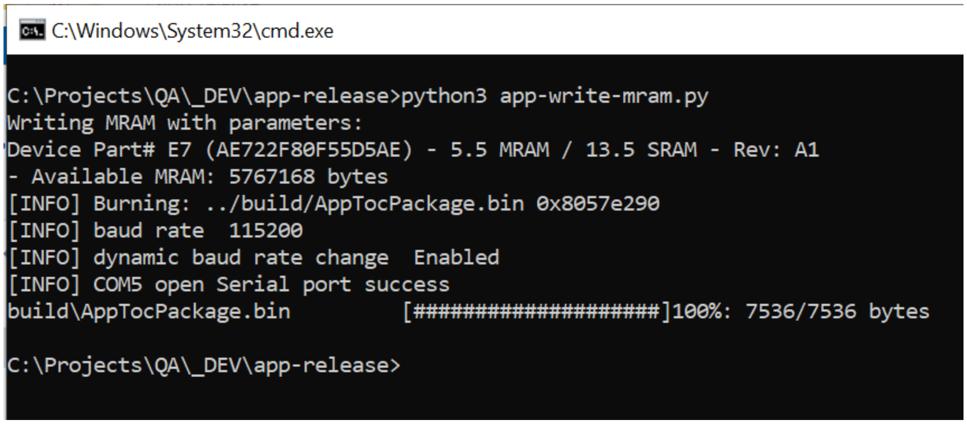
From the Windows Command prompt;

```
$ python3 app-write-mram.py  (-h for help)
```



```
$ python3 app-write-mram.py -h
usage: app-write-mram.py [-h] [-a AUTH_HOST] [-d] [-b BAUDRATE] [-e ERASE] [-i IMAGES | -S] [-s] [-x] [-V] [-v]

NVM Burner for Application TOC Package

optional arguments:
  -h, --help            show this help message and exit
  -a AUTH_HOST, --auth_host AUTH_HOST
  -d, --discover        COM port discovery
  -b BAUDRATE, --baudrate BAUDRATE
                        serial port baud rate
  -e ERASE, --erase ERASE
                        ERASE [APP | <start address> <size> [<pattern>] ]
  -i IMAGES, --images IMAGES
                        images list to burn into NVM ("/path/image1.bin 0x80001000 /path/image2.bin 0x80003000")
  -S, --skip            write ATOC only - skip user managed images
  -s, --switch          dynamic baud rate switch toggle, default=on
  -x, --exit            exit on NAK
  -V, --version         Display Version Number
  -v, --verbose         verbosity mode
```

Before running this utility, be sure the CPU board is powered on and the SEUART is connected to the PC. Open Windows Command prompt and change directory to the release directory. Call the script as the example shows.



C:\Windows\System32\cmd.exe

```
C:\Projects\QA\_DEV\app-release>python3 app-write-mram.py
Writing MRAM with parameters:
Device Part# E7 (AE722F80F55D5AE) - 5.5 MRAM / 13.5 SRAM - Rev: A1
- Available MRAM: 5767168 bytes
[INFO] Burning: ../build/AppTocPackage.bin 0x8057e290
[INFO] baud rate  115200
[INFO] dynamic baud rate change  Enabled
[INFO] COM5 open Serial port success
build\AppTocPackage.bin         [####################]100%: 7536/7536 bytes

C:\Projects\QA\_DEV\app-release>
```

The first time this command is executed it will prompt the user for the Serial Communications port name. This information is stored in a local file and will be read again when this command is executed.

You can override this using the -d (discovery) option and you will be prompted again for the port name.

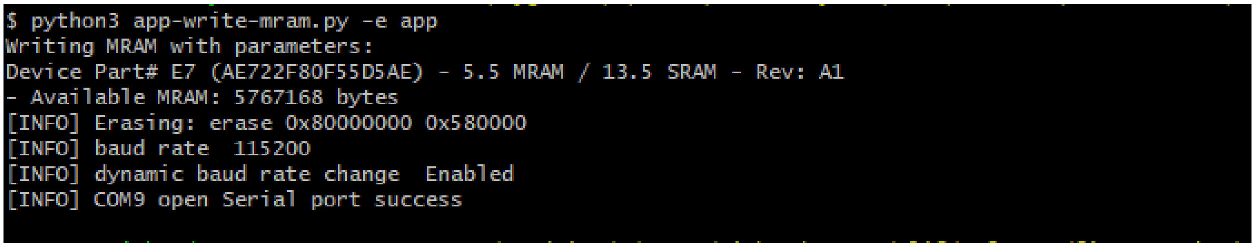
There is a verbose mode (-v) which will print all the communications between the host and the target. During this mode, the progress bar is suppressed.

Erase option

Erase option will allow, via ISP, the contents of the Application MRAM to be erased. Erasing means that the MRAM location is written with zeros. The write is verified.

Erasing all the application MRAM

```
$ python3 app-write-mram.py -e app
```



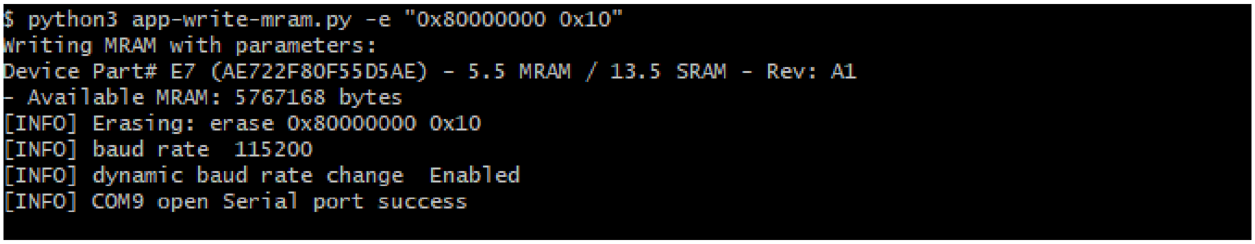
```
$ python3 app-write-mram.py -e app
Writing MRAM with parameters:
Device Part# E7 (AE722F80F55D5AE) - 5.5 MRAM / 13.5 SRAM - Rev: A1
- Available MRAM: 5767168 bytes
[INFO] Erasing: erase 0x80000000 0x580000
[INFO] baud rate  115200
[INFO] dynamic baud rate change  Enabled
[INFO] COM9 open Serial port success
```

Depending on the size of the Application MRAM this can take a few seconds to complete, do not panic the tool will exit when the operation completes, or an error is detected.

NOTE: The pattern written is always zeros and cannot be changed.

Erasing a specific address of application MRAM

The user can specify a specific address and length to erase

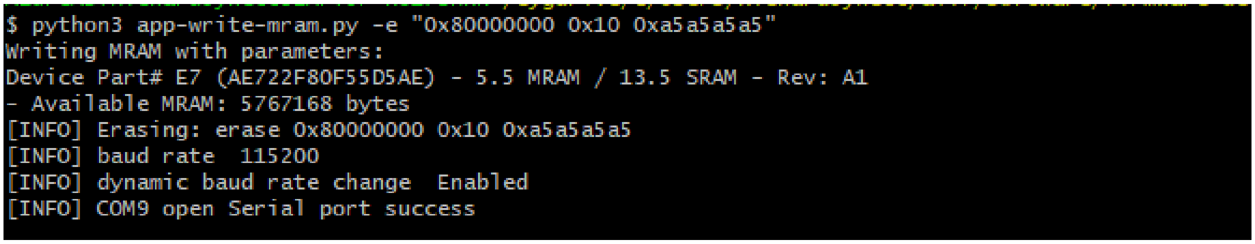


```
$ python3 app-write-mram.py -e "0x80000000 0x10"
Writing MRAM with parameters:
Device Part# E7 (AE722F80F55D5AE) - 5.5 MRAM / 13.5 SRAM - Rev: A1
- Available MRAM: 5767168 bytes
[INFO] Erasing: erase 0x80000000 0x10
[INFO] baud rate  115200
[INFO] dynamic baud rate change  Enabled
[INFO] COM9 open Serial port success
```

This will erase from the start of MRAM for 16 Bytes, it will be erased with zeros.

Erasing a specific address of application MRAM with a pattern

The user can specify a specific address and length to erase along with a pattern to be written

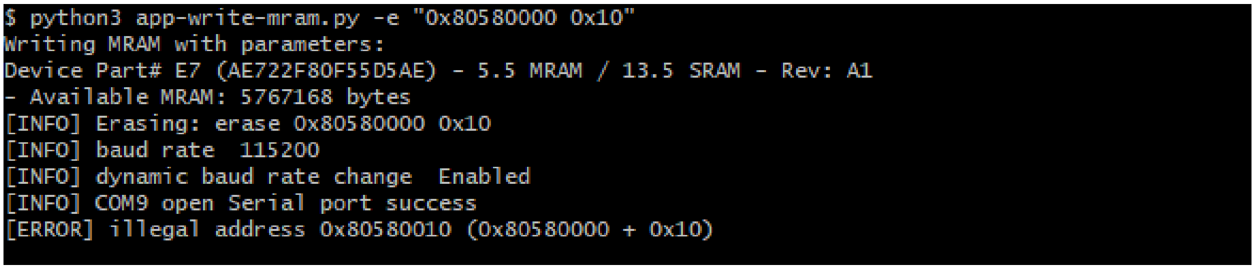


```
$ python3 app-write-mram.py -e "0x80000000 0x10 0xa5a5a5a5"
Writing MRAM with parameters:
Device Part# E7 (AE722F80F55D5AE) - 5.5 MRAM / 13.5 SRAM - Rev: A1
- Available MRAM: 5767168 bytes
[INFO] Erasing: erase 0x80000000 0x10 0xa5a5a5a5
[INFO] baud rate  115200
[INFO] dynamic baud rate change  Enabled
[INFO] COM9 open Serial port success
```

In this example, the pattern 0xa5a5a5a5 will be written to the <address> for <length> bytes.

Erasing a specific address of MRAM that is illegal

If the user specifies a region of MRAM that is not allowed to be accessed



```
$ python3 app-write-mram.py -e "0x80580000 0x10"
Writing MRAM with parameters:
Device Part# E7 (AE722F80F55D5AE) - 5.5 MRAM / 13.5 SRAM - Rev: A1
- Available MRAM: 5767168 bytes
[INFO] Erasing: erase 0x80580000 0x10
[INFO] baud rate  115200
[INFO] dynamic baud rate change  Enabled
[INFO] COM9 open Serial port success
[ERROR] illegal address 0x80580010 (0x80580000 + 0x10)
```

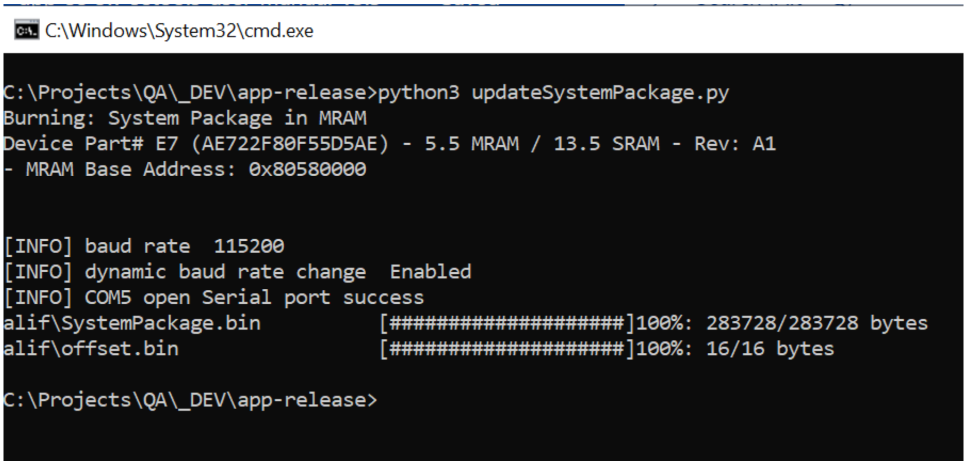
In this example, an error is flagged that the address is illegal

updateSystemPackage.py

From the Windows Command prompt;

```
$ python3 updateSystemPackage.py  (-h for help)
```

Before running this utility, be sure the CPU board is powered on and the SEUART is connected to the PC. Open Windows Command prompt and change directory to the release directory. Call the script as the example shows.



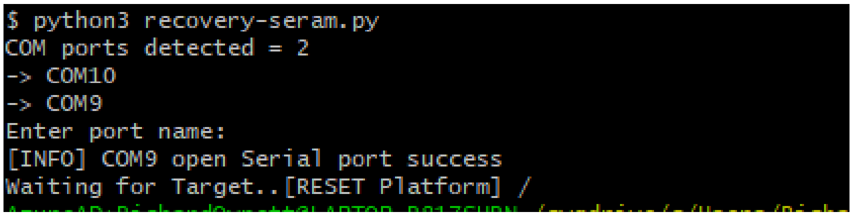
C:\Windows\System32\cmd.exe

```
C:\Projects\QA\_DEV\app-release>python3 updateSystemPackage.py
Burning: System Package in MRAM
Device Part# E7 (AE722F80F55D5AE) - 5.5 MRAM / 13.5 SRAM - Rev: A1
- MRAM Base Address: 0x80580000


[INFO] baud rate  115200
[INFO] dynamic baud rate change  Enabled
[INFO] COM5 open Serial port success
alif\SystemPackage.bin          [####################]100%: 283728/283728 bytes
alif\offset.bin                 [####################]100%: 16/16 bytes

C:\Projects\QA\_DEV\app-release>
```

SERAM Recovery Mode

To allow for recovery of MRAM use the following command

```
$ python3 recover-seram.py
```



```
$ python3 recovery-seram.py
COM ports detected = 2
-> COM10
-> COM9
Enter port name:
[INFO] COM9 open Serial port success
Waiting for Target..[RESET Platform] /
```

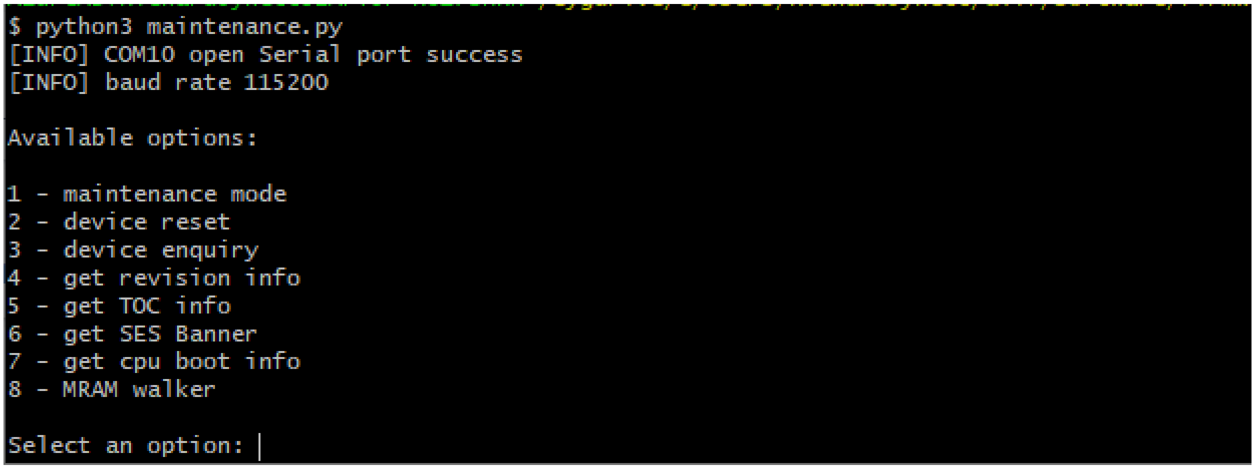
Once running, you can reset your target board. This process will cause the tool to exit as communication has occurred with the Target. The
Target is now able to re-program ATOC.

maintenance.py

Maintenance mode allows for the recovery of MRAM executable images. NOTE: This is like recovery-seram.py but allows more commands to be executed

Running the Tool

```
$ python3 maintenance.py (-h for help)
```



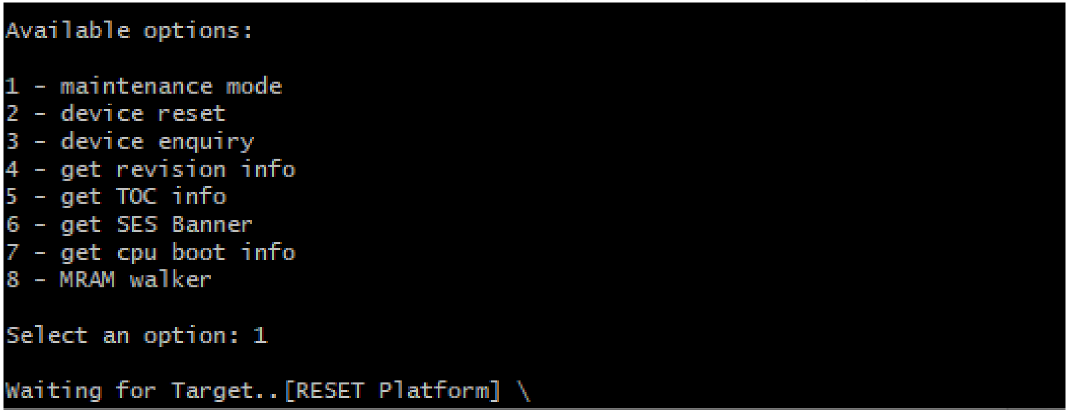
```
$ python3 maintenance.py
[INFO] COM10 open Serial port success
[INFO] baud rate 115200

Available options:

1 - maintenance mode
2 - device reset
3 - device enquiry
4 - get revision info
5 - get TOC info
6 - get SES Banner
7 - get cpu boot info
8 - MRAM walker

Select an option: |
```

The command presents several options including maintenance mode. Select an option number from the menu.

To exit the tool just press [ENTER] at the option selection.

Maintenance Mode

Maintenance mode will wait until the user has pressed RESET on the target board.



```
Available options:

1 - maintenance mode
2 - device reset
3 - device enquiry
4 - get revision info
5 - get TOC info
6 - get SES Banner
7 - get cpu boot info
8 - MRAM walker

Select an option: 1

Waiting for Target..[RESET Platform] \
```

Once RESET has been pressed the tool will drop back to the Available options menu. NOTE: This mode skips the TOC processing section of SES, if you use the get TOC info option it will return no data as no TOCs have been loaded.

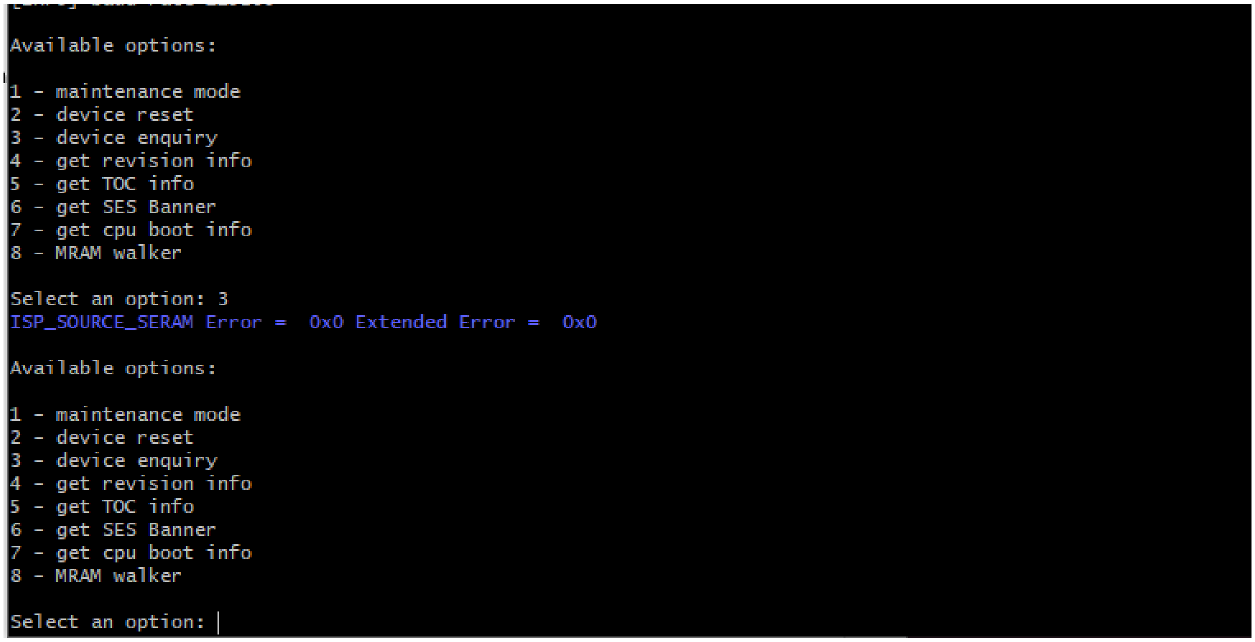
To exit the tool, just press [ENTER]

Device Reset

Issues a Reset command to the board.

Device Enquiry

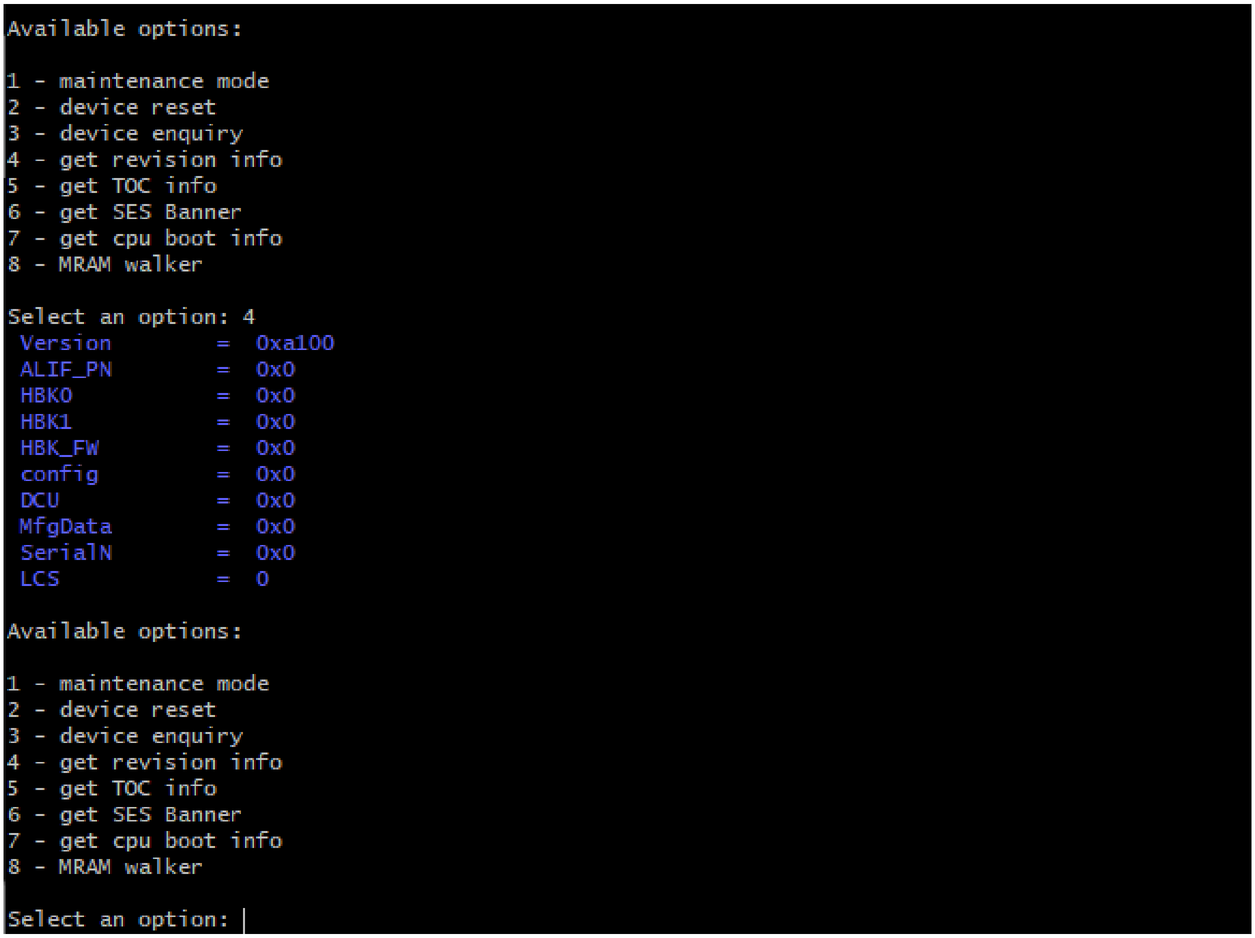
Returns if connected to SEROM or SES. Any errors are reported otherwise 0x0 (Success) is returned.



```
Available options:

1 - maintenance mode
2 - device reset
3 - device enquiry
4 - get revision info
5 - get TOC info
6 - get SES Banner
7 - get cpu boot info
8 - MRAM walker

Select an option: 3
ISP_SOURCE_SERAM Error =  0x0 Extended Error =  0x0

Available options:

1 - maintenance mode
2 - device reset
3 - device enquiry
4 - get revision info
5 - get TOC info
6 - get SES Banner
7 - get cpu boot info
8 - MRAM walker

Select an option: |
```

Get Revision Information

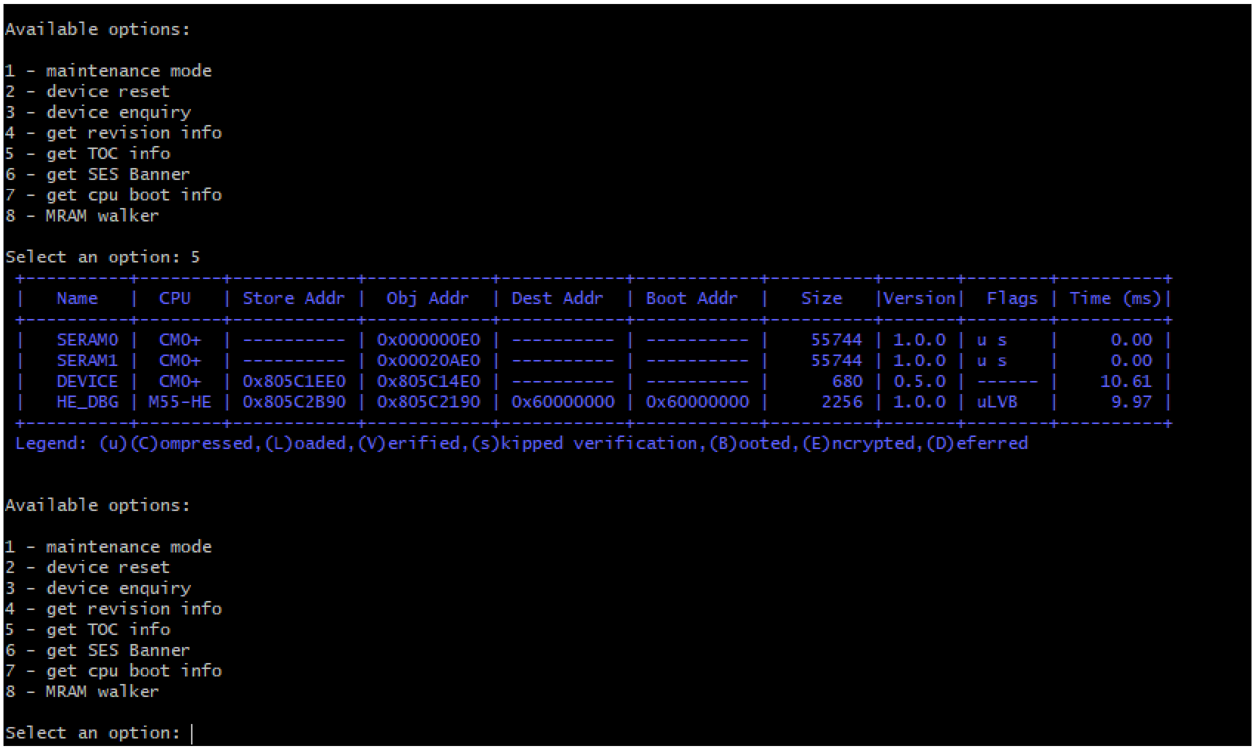
Returns device specific information such as the SOC ID, for REV_A1 this will be 0xA100.



```
Available options:

1 - maintenance mode
2 - device reset
3 - device enquiry
4 - get revision info
5 - get TOC info
6 - get SES Banner
7 - get cpu boot info
8 - MRAM walker

Select an option: 4
 Version         =  0xa100
 ALIF_PN         =  0x0
 HBK0            =  0x0
 HBK1            =  0x0
 HBK_FW          =  0x0
 config          =  0x0
 DCU             =  0x0
 MfgData         =  0x0
 SerialN         =  0x0
 LCS             =  0

Available options:

1 - maintenance mode
2 - device reset
3 - device enquiry
4 - get revision info
5 - get TOC info
6 - get SES Banner
7 - get cpu boot info
8 - MRAM walker

Select an option: 
```

Get Table of contents information

This will return the TOC contents in MRAM along with the boot status of the Application CPUs.



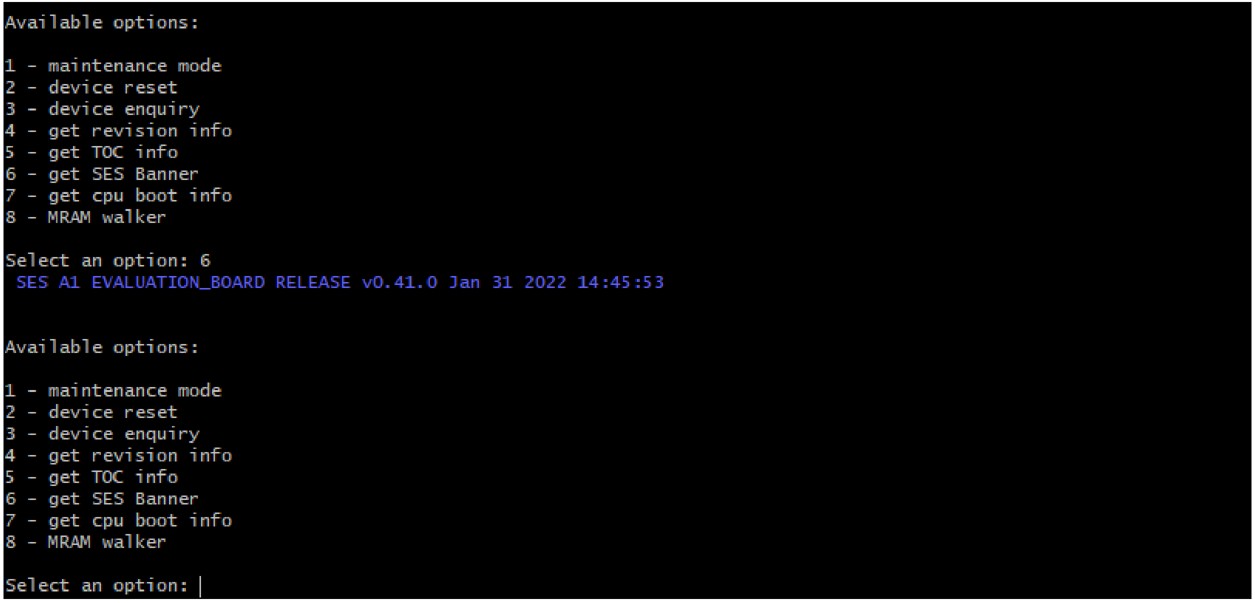
```
Available options:

1 - maintenance mode
2 - device reset
3 - device enquiry
4 - get revision info
5 - get TOC info
6 - get SES Banner
7 - get cpu boot info
8 - MRAM walker

Select an option: 5
 +----------+--------+------------+------------+------------+------------+------------+-------+--------+----------+
 |   Name   |  CPU   | Store Addr |  Obj Addr  | Dest Addr  | Boot Addr  |    Size    |Version| Flags  | Time (ms)|
 +----------+--------+------------+------------+------------+------------+------------+-------+--------+----------+
 |   SERAM0 |  CM0+  | ---------- | 0x000000E0 | ---------- | ---------- |      55744 | 1.0.0 | u s    |     0.00 |
 |   SERAM1 |  CM0+  | ---------- | 0x00020AE0 | ---------- | ---------- |      55744 | 1.0.0 | u s    |     0.00 |
 |   DEVICE |  CM0+  | 0x805C1EE0 | 0x805C14E0 | ---------- | ---------- |        680 | 0.5.0 | ------ |    10.61 |
 |   HE_DBG | M55-HE | 0x805C2B90 | 0x805C2190 | 0x60000000 | 0x60000000 |       2256 | 1.0.0 | uLVB   |     9.97 |
 +----------+--------+------------+------------+------------+------------+------------+-------+--------+----------+
 Legend: (u)(C)ompressed,(L)oaded,(V)erified,(s)kipped verification,(B)ooted,(E)ncrypted,(D)eferred


Available options:

1 - maintenance mode
2 - device reset
3 - device enquiry
4 - get revision info
5 - get TOC info
6 - get SES Banner
7 - get cpu boot info
8 - MRAM walker

Select an option: |
```

Legend Reference

Name	Name of entry from JSON file.	
CPU	Which CPU is this TOC referencing.	
Store Address	MRAM location	
Object address	MRAM TOC address	
Destination address	Copy address for object (if relevant).	
Boot Address	Boot address	
Size	Size of TOC entry	
Version	Version id from JSON file	
Flags	Directives for the TOC object	
	u	Uncompressed image
	C	Compressed Image
	L	Image is loaded from MRAM
	V	Image is verified
	s	Image is Skipped
	B	Image has been Booted
	E	Image is Encrypted
	D	Image is Deferred, will be Loaded / Booted at a later time

SES Banner

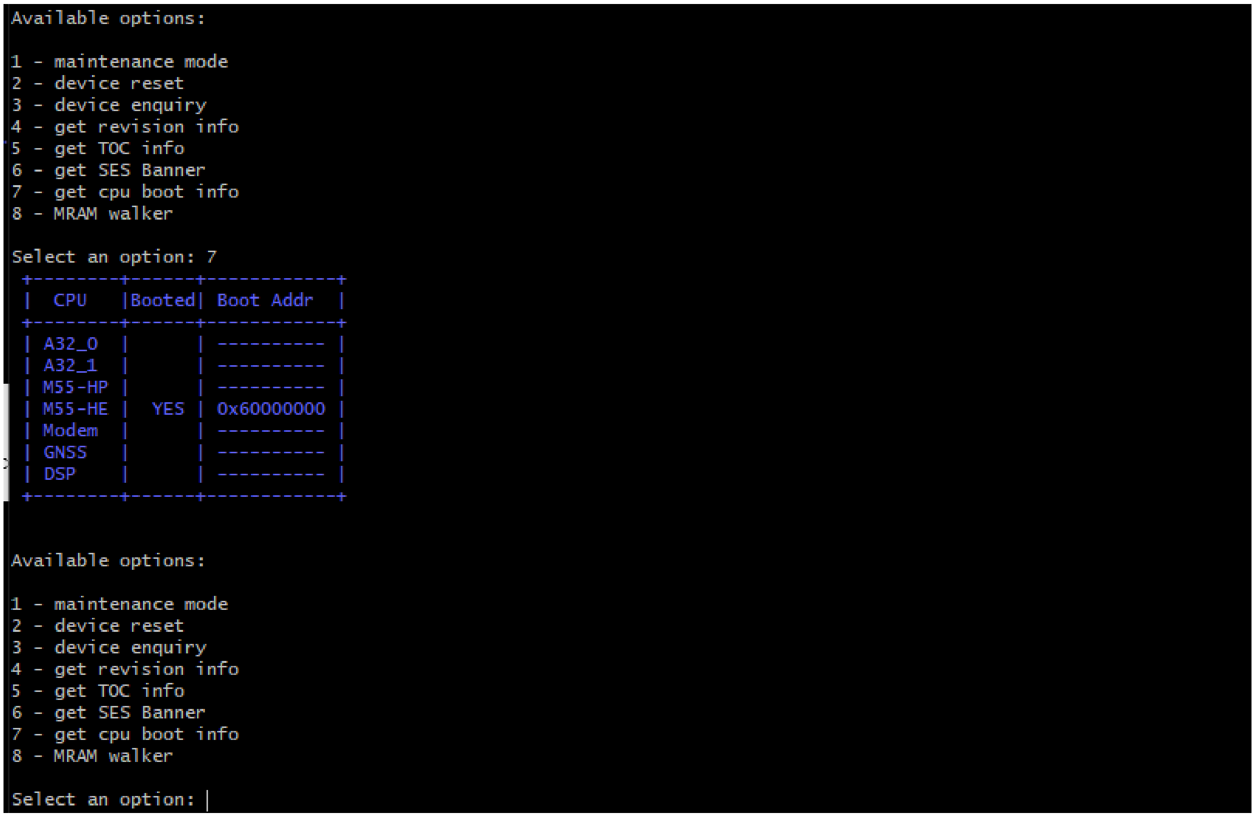
This returns the version string for the SES image



```
Available options:

1 - maintenance mode
2 - device reset
3 - device enquiry
4 - get revision info
5 - get TOC info
6 - get SES Banner
7 - get cpu boot info
8 - MRAM walker

Select an option: 6
 SES A1 EVALUATION_BOARD RELEASE v0.41.0 Jan 31 2022 14:45:53


Available options:

1 - maintenance mode
2 - device reset
3 - device enquiry
4 - get revision info
5 - get TOC info
6 - get SES Banner
7 - get cpu boot info
8 - MRAM walker

Select an option: |
```

Get CPU boot information

This returns the status of the CPU cores.

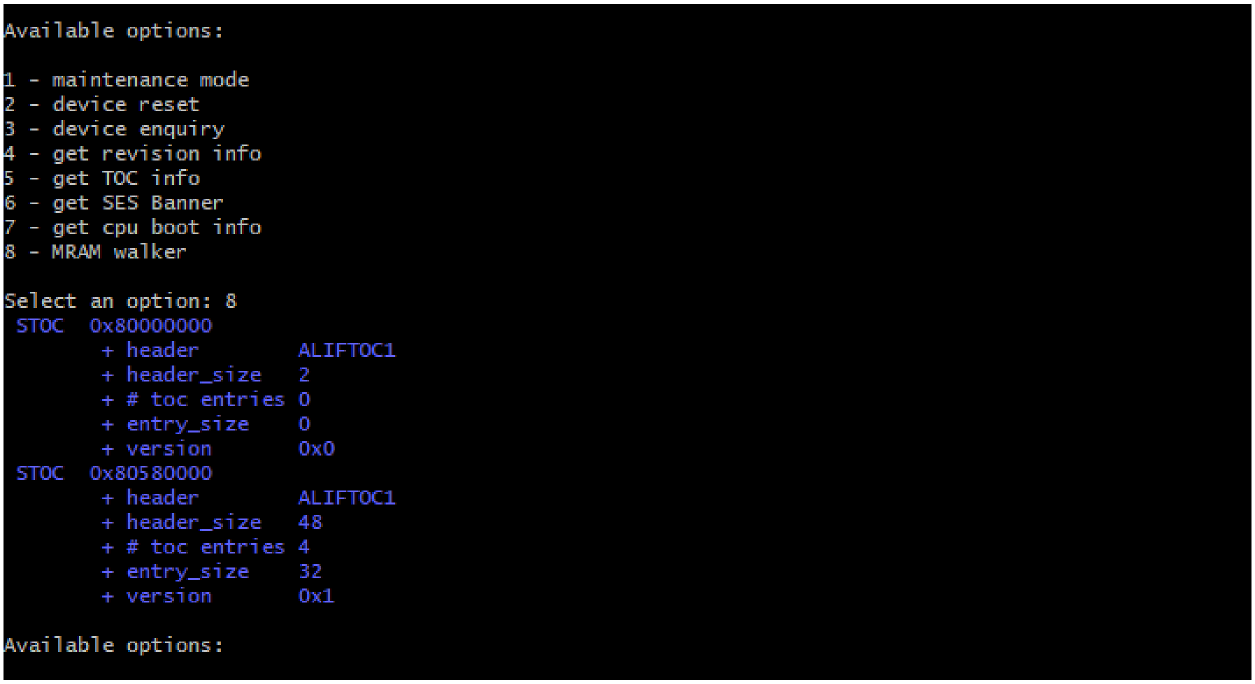


```
Available options:

1 - maintenance mode
2 - device reset
3 - device enquiry
4 - get revision info
5 - get TOC info
6 - get SES Banner
7 - get cpu boot info
8 - MRAM walker

Select an option: 7
 +--------+------+------------+
 |  CPU   |Booted| Boot Addr  |
 +--------+------+------------+
 | A32_0  |      | ---------- |
 | A32_1  |      | ---------- |
 | M55-HP |      | ---------- |
 | M55-HE |  YES | 0x60000000 |
 | Modem  |      | ---------- |
 | GNSS   |      | ---------- |
 | DSP    |      | ---------- |
 +--------+------+------------+


Available options:

1 - maintenance mode
2 - device reset
3 - device enquiry
4 - get revision info
5 - get TOC info
6 - get SES Banner
7 - get cpu boot info
8 - MRAM walker

Select an option: |
```

This example which CPUs are currently booted.

MRAM Walker

This walks the entire MRAM memory looking for Table of Contents objects

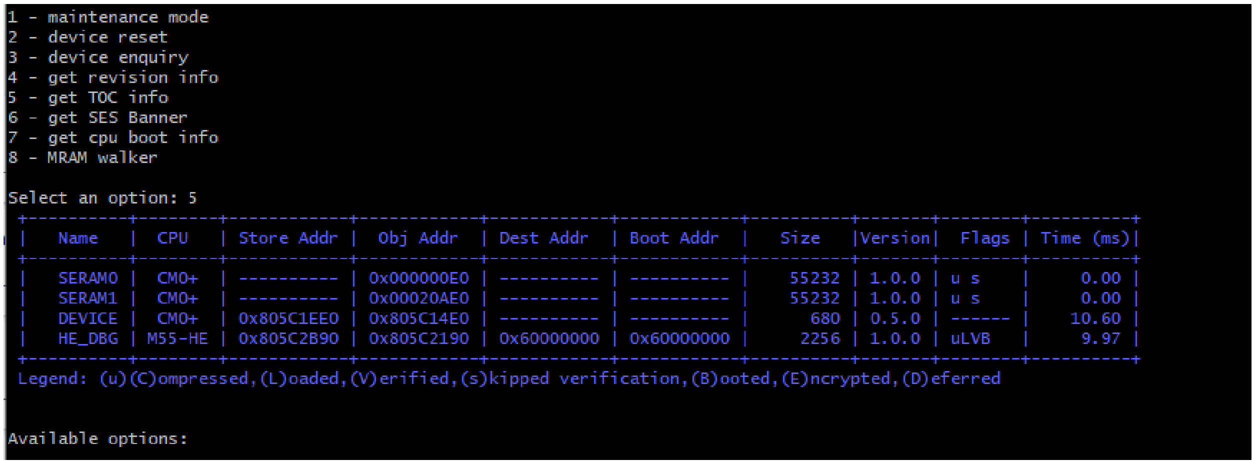


```
Available options:

1 - maintenance mode
2 - device reset
3 - device enquiry
4 - get revision info
5 - get TOC info
6 - get SES Banner
7 - get cpu boot info
8 - MRAM walker

Select an option: 8
 STOC  0x80000000
       + header         ALIFTOC1
       + header_size    2
       + # toc entries 0
       + entry_size     0
       + version        0x0
 STOC  0x80580000
       + header         ALIFTOC1
       + header_size    48
       + # toc entries 4
       + entry_size     32
       + version        0x1

Available options:
```

In this example (For REV_A0), two System TOC objects were found.

The first is the initial boot object, note that the # of toc entries is 0.

The second shows the # of toc entries is 4 which equates to



```
1 - maintenance mode
2 - device reset
3 - device enquiry
4 - get revision info
5 - get TOC info
6 - get SES Banner
7 - get cpu boot info
8 - MRAM walker

Select an option: 5
 +----------+--------+------------+------------+------------+------------+----------+-------+--------+----------+
 |   Name   |  CPU   | Store Addr |  Obj Addr  | Dest Addr  | Boot Addr  |   Size   |Version| Flags  | Time (ms)|
 +----------+--------+------------+------------+------------+------------+----------+-------+--------+----------+
 |   SERAM0 |  CM0+  | ---------- | 0x000000E0 | ---------- | ---------- |    55232 | 1.0.0 | u s    |     0.00 |
 |   SERAM1 |  CM0+  | ---------- | 0x00020AE0 | ---------- | ---------- |    55232 | 1.0.0 | u s    |     0.00 |
 |   DEVICE |  CM0+  | 0x805C1EE0 | 0x805C14E0 | ---------- | ---------- |      680 | 0.5.0 | ------ |    10.60 |
 |   HE_DBG | M55-HE | 0x805C2B90 | 0x805C2190 | 0x60000000 | 0x60000000 |     2256 | 1.0.0 | uLVB   |     9.97 |
 +----------+--------+------------+------------+------------+------------+----------+-------+--------+----------+
 Legend: (u)(C)ompressed,(L)oaded,(V)erified,(s)kipped verification,(B)ooted,(E)ncrypted,(D)eferred


Available options:
```

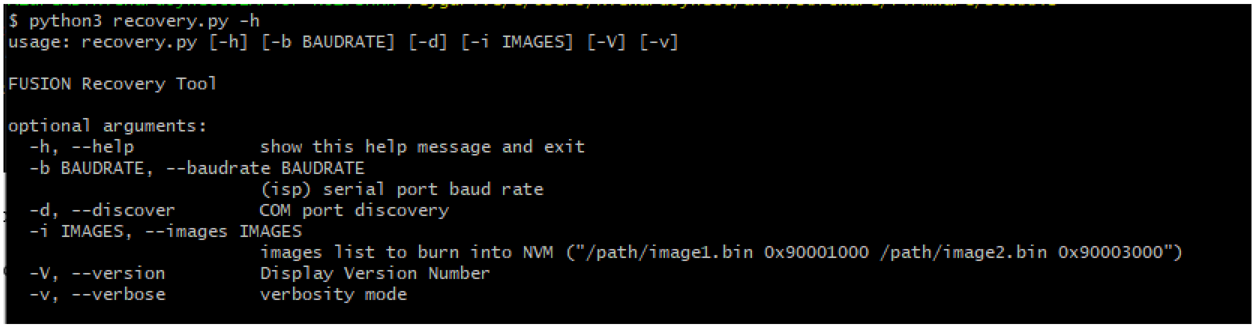
SEROM Recovery Mode

REV_A1 devices support a limited subset of ISP commands built into the ROM.

If the NVM contains no viable SERAM image to load, SEROM will enter ISP mode to allow recovery. Recovery will take an SERAM image and write it to the NVM.

There is a separate command to achieve this

```
$ python3 recovery.py -h
```



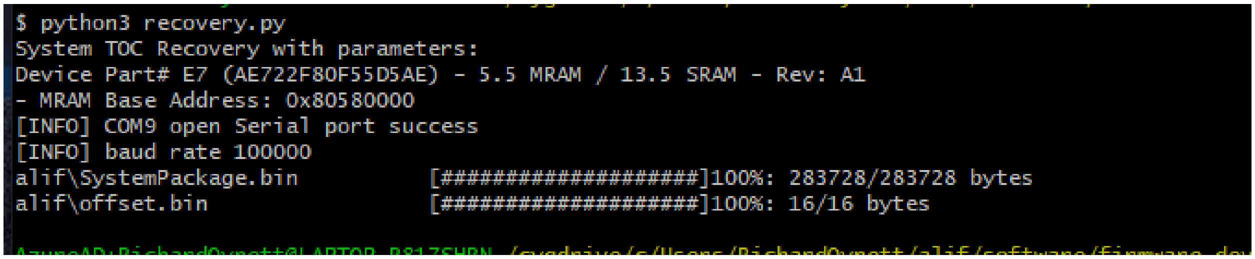
```
$ python3 recovery.py -h
usage: recovery.py [-h] [-b BAUDRATE] [-d] [-i IMAGES] [-V] [-v]

FUSION Recovery Tool

optional arguments:
  -h, --help            show this help message and exit
  -b BAUDRATE, --baudrate BAUDRATE
                        (isp) serial port baud rate
  -d, --discover        COM port discovery
  -i IMAGES, --images IMAGES
                        images list to burn into NVM ("/path/image1.bin 0x90001000 /path/image2.bin 0x90003000")
  -V, --version         Display Version Number
  -v, --verbose         verbosity mode
```

It uses the specific ISP protocol supported by SEROM.

SEROM Recovery Mode (REV_A1)

If SEROM fails to find a valid (or no) SERAM image in the NVM it will enter ISP mode.

```
        $ python3 recovery.py
```



```
$ python3 recovery.py
System TOC Recovery with parameters:
Device Part# E7 (AE722F80F55D5AE) - 5.5 MRAM / 13.5 SRAM - Rev: A1
- MRAM Base Address: 0x80580000
[INFO] COM9 open Serial port success
[INFO] baud rate 100000
alif\SystemPackage.bin          [####################]100%: 283728/283728 bytes
alif\offset.bin                 [####################]100%: 16/16 bytes
```

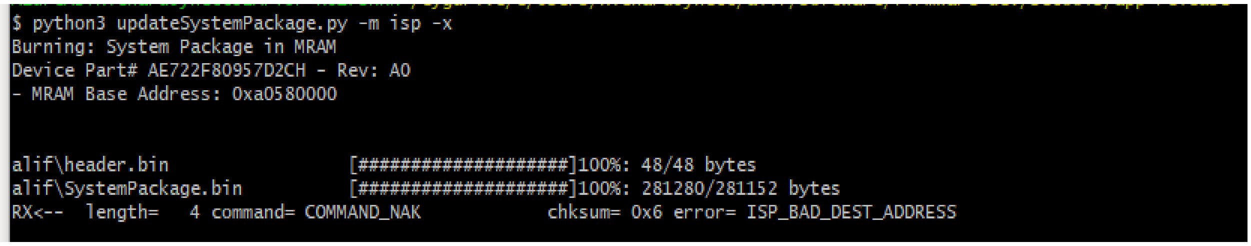
This command will allow the user to restore the System TOC package in NVM.

NOTE: The baud rate is different for SEROM. The recovery tool automatically switches to the correct baud rate for this device. The ‘-b’ baud rate option is still supported.

NACK Error Handling

The ISP protocol uses an Acknowledge (ACK) / Not Acknowledge (NACK) protocol scheme. If NACK is seen this is followed by an Error code.

If a NACK packet is seen it will always be printed and the tool will EXIT.

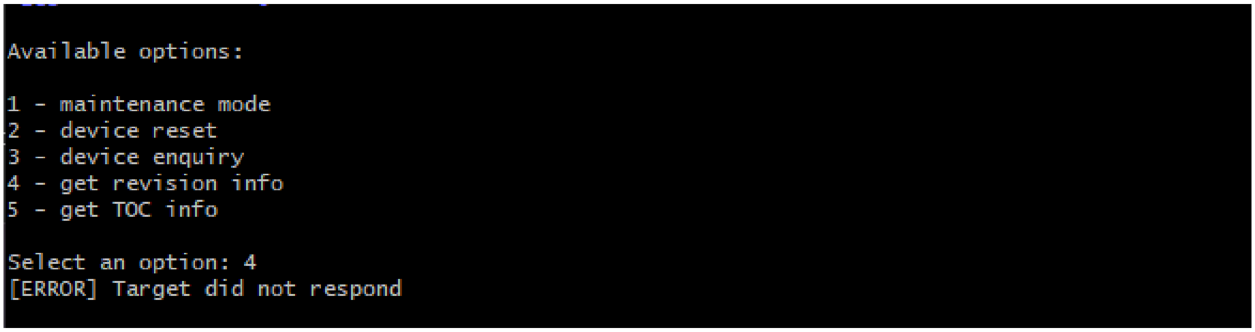


```
$ python3 updateSystemPackage.py -m isp -x
Burning: System Package in MRAM
Device Part# AE722F80957D2CH - Rev: A0
- MRAM Base Address: 0xa0580000


alif\header.bin                [####################]100%: 48/48 bytes
alif\SystemPackage.bin         [####################]100%: 281280/281152 bytes
RX<--  length=   4 command= COMMAND_NAK            chksum= 0x6 error= ISP_BAD_DEST_ADDRESS
```

The -x option will cause the tool to continue and not Exit if a NACK is seen.

TIMEOUT Error Handling

An ISP session requires an initial ISP START command sent to the Target to put it into ISP mode. If this command fails, then the ISP session will not occur and an Error is reported that the Target did not respond



```
Available options:

1 - maintenance mode
2 - device reset
3 - device enquiry
4 - get revision info
5 - get TOC info

Select an option: 4
[ERROR] Target did not respond
```

This Error will exit the program.

Possible causes for this Error:

- Incorrect baud rate setting
- Target is not powered

ISP Commands

Executing ISP commands

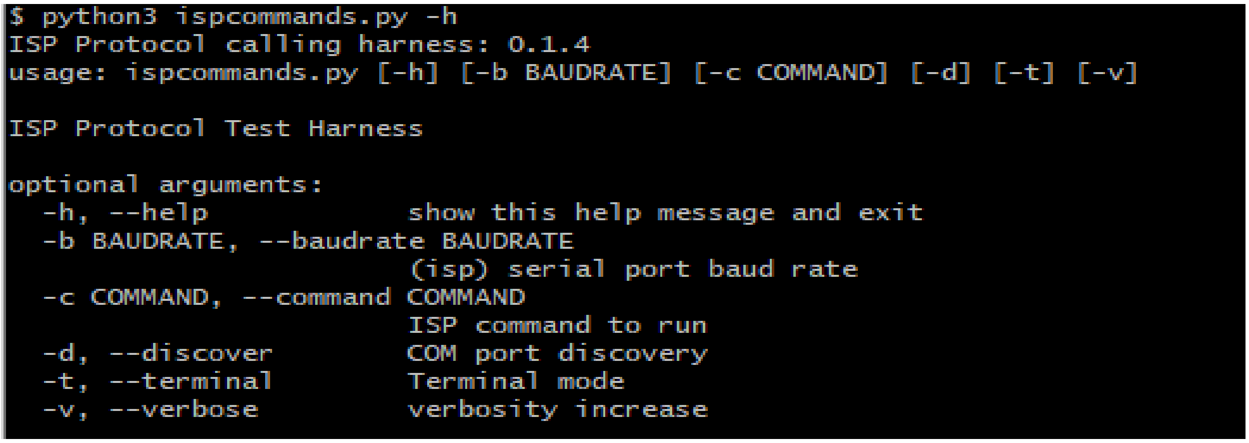
Each ISP protocol can be executed individually through an ISP test harness.

This is in setools/isp/ispcommands.py.

```
$ cd isp
$ python3 ispcommands.py -h
```

The script detects and displays the available COM ports and expects the user to enter which one to use.
The default COM port is defined by the value of 'COM_PORT_DEFAULT' in file serialport.py.
You can modify that value and set the correct COM port for your environment, so you do not have to enter it each time the script is run (just hit Enter on the prompt ‘Enter port name:’).

The following output show be displayed:



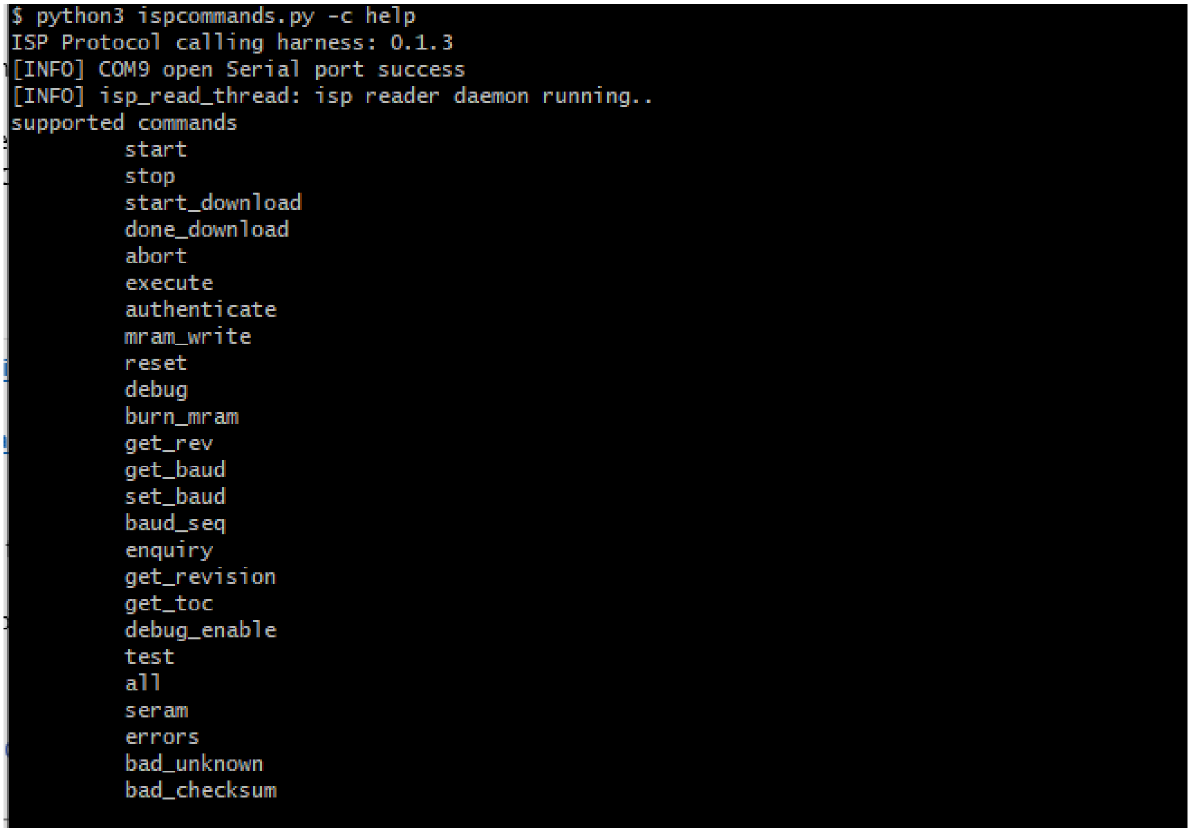
```
$ python3 ispcommands.py -h
ISP Protocol calling harness: 0.1.4
usage: ispcommands.py [-h] [-b BAUDRATE] [-c COMMAND] [-d] [-t] [-v]

ISP Protocol Test Harness

optional arguments:
  -h, --help            show this help message and exit
  -b BAUDRATE, --baudrate BAUDRATE
                        (isp) serial port baud rate
  -c COMMAND, --command COMMAND
                        ISP command to run
  -d, --discover        COM port discovery
  -t, --terminal        Terminal mode
  -v, --verbose         verbosity increase
```

-h	Help menu
-c <command>	ISP Command execution
-d	ISP Discovery
-b <baudrate>	Specify a new baudrate
-t	Terminal
-v	Verbosity level

Each <command> can be called, see later.

ISP Command Help

```
$ cd isp
$ python3 ispcommands.py -c help
```



```
$ python3 ispcommands.py -c help
ISP Protocol calling harness: 0.1.3
[INFO] COM9 open Serial port success
[INFO] isp_read_thread: isp reader daemon running..
supported commands
         start
         stop
         start_download
         done_download
         abort
         execute
         authenticate
         mram_write
         reset
         debug
         burn_mram
         get_rev
         get_baud
         set_baud
         baud_seq
         enquiry
         get_revision
         get_toc
         debug_enable
         test
         all
         seram
         errors
         bad_unknown
         bad_checksum
```

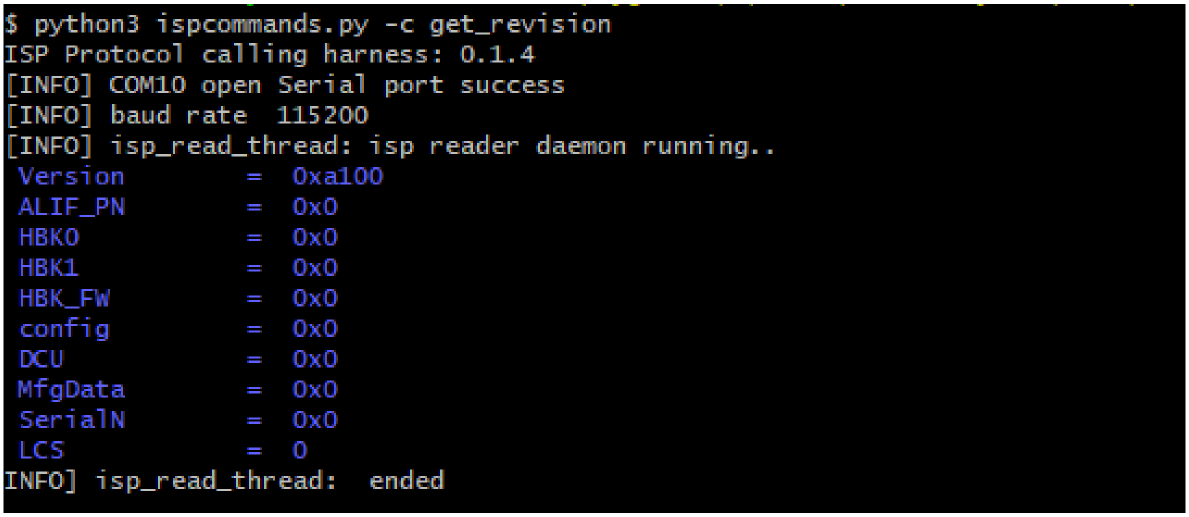
The help option shows all the available ISP commands.

Some of these options will issue the ISP protocol command (-c <command>) to the target and are used for testing purposes.

bad_unknown and bad_checksum are tests for the protocol. Use the -v option to see the packet traffic.

Get Revision

This command requests the device revision details.

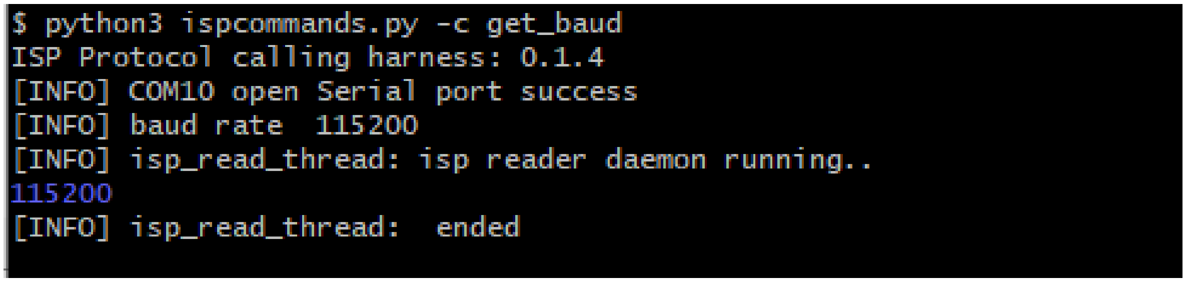


```
$ python3 ispcommands.py -c get_revision
ISP Protocol calling harness: 0.1.4
[INFO] COM10 open Serial port success
[INFO] baud rate  115200
[INFO] isp_read_thread: isp reader daemon running..
 Version        =  0xa100
 ALIF_PN        =  0x0
 HBK0           =  0x0
 HBK1           =  0x0
 HBK_FW         =  0x0
 config         =  0x0
 DCU            =  0x0
 MfgData        =  0x0
 SerialN        =  0x0
 LCS            =  0
INFO] isp_read_thread:  ended
```

NOTE: REV_A0 has no device revision register, in this case it will return the SERAM revision.

Get Baud Rate

This command requests the device current baud rate



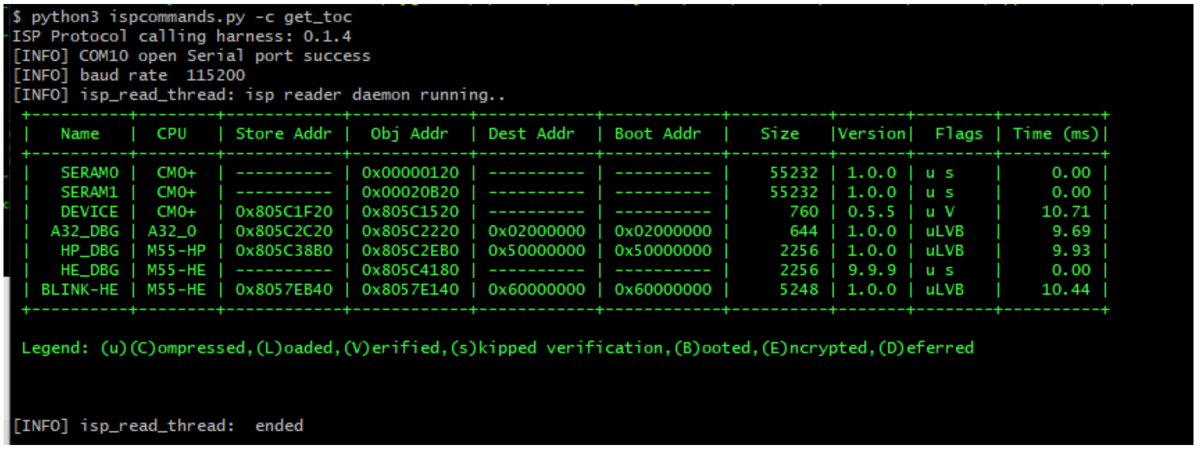
```
$ python3 ispcommands.py -c get_baud
ISP Protocol calling harness: 0.1.4
[INFO] COM10 open Serial port success
[INFO] baud rate  115200
[INFO] isp_read_thread: isp reader daemon running..
115200
[INFO] isp_read_thread:  ended
```

Get TOC Information

This command retrieves the Table of Contents information.

Shows the MRAM contents and boot status.

```
$ cd isp
$ python3 ispcommands.py -c get_toc
```

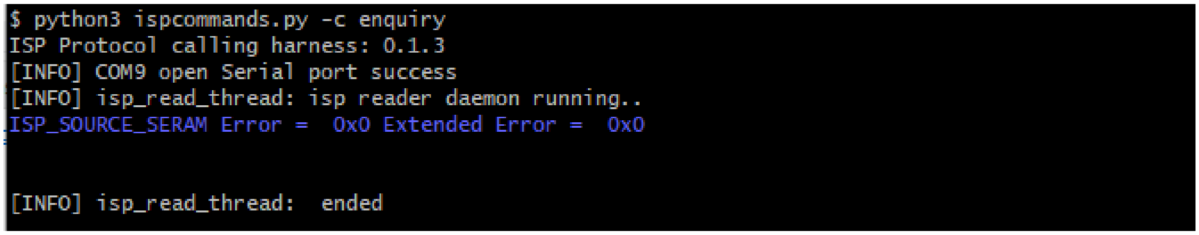


```
$ python3 ispcommands.py -c get_toc
ISP Protocol calling harness: 0.1.4
[INFO] COM10 open Serial port success
[INFO] baud rate  115200
[INFO] isp_read_thread: isp reader daemon running..
 +----------+--------+------------+------------+------------+------------+----------+-------+--------+----------+
 |   Name   |  CPU   | Store Addr |  Obj Addr  | Dest Addr  | Boot Addr  |   Size   |Version| Flags  | Time (ms)|
 +----------+--------+------------+------------+------------+------------+----------+-------+--------+----------+
 |   SERAM0 |  CM0+  | ---------- | 0x00000120 | ---------- | ---------- |    55232 | 1.0.0 | u s    |     0.00 |
 |   SERAM1 |  CM0+  | ---------- | 0x00020B20 | ---------- | ---------- |    55232 | 1.0.0 | u s    |     0.00 |
 |   DEVICE |  CM0+  | 0x805C1F20 | 0x805C1520 | ---------- | ---------- |      760 | 0.5.5 | u V    |    10.71 |
 |  A32_DBG | A32_0  | 0x805C2C20 | 0x805C2220 | 0x02000000 | 0x02000000 |      644 | 1.0.0 | uLVB   |     9.69 |
 |   HP_DBG | M55-HP | 0x805C38B0 | 0x805C2EB0 | 0x50000000 | 0x50000000 |     2256 | 1.0.0 | uLVB   |     9.93 |
 |   HE_DBG | M55-HE | ---------- | 0x805C4180 | ---------- | ---------- |     2256 | 9.9.9 | u s    |     0.00 |
 | BLINK-HE | M55-HE | 0x8057EB40 | 0x8057E140 | 0x60000000 | 0x60000000 |     5248 | 1.0.0 | uLVB   |    10.44 |
 +----------+--------+------------+------------+------------+------------+----------+-------+--------+----------+

 Legend: (u)(C)ompressed,(L)oaded,(V)erified,(s)kipped verification,(B)ooted,(E)ncrypted,(D)eferred


[INFO] isp_read_thread:  ended
```

Enquiry

This command performs an enquiry of SERAM / SEROM. This protocol is used by other tools such as recovery mode for SEROM.

```
$ cd isp
$ python3 ispcommands.py -c enquiry
```



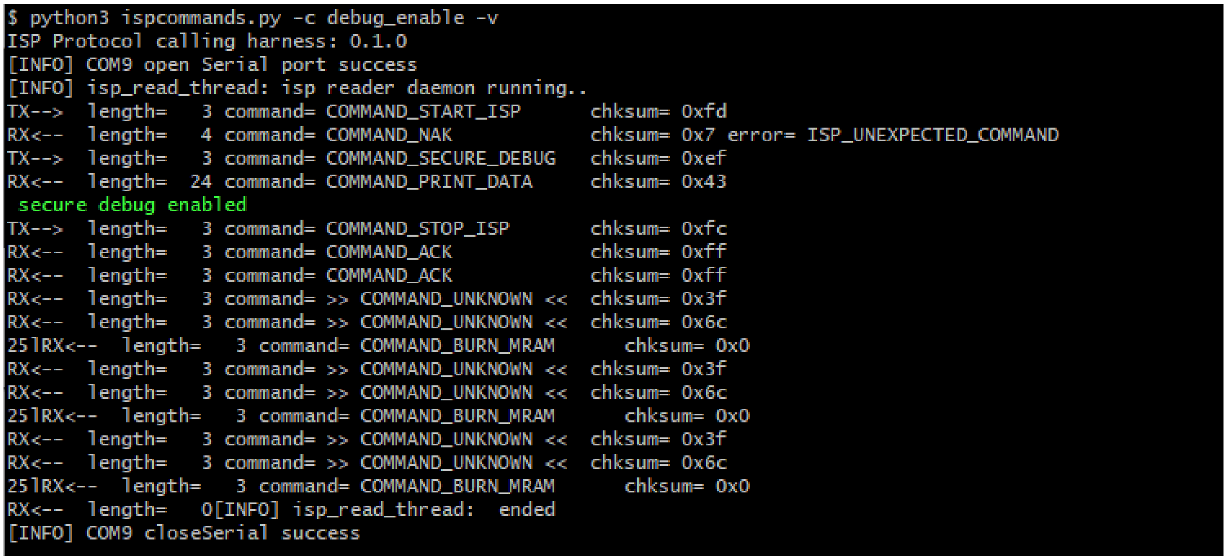
```
$ python3 ispcommands.py -c enquiry
ISP Protocol calling harness: 0.1.3
[INFO] COM9 open Serial port success
[INFO] isp_read_thread: isp reader daemon running..
ISP_SOURCE_SERAM Error =  0x0 Extended Error =  0x0


[INFO] isp_read_thread:  ended
```

In this case this is returning the source as SERAM and reports Error as 0x0. Extended Error is always 0x0 for SERAM.

In the case of SEROM, there is also an Extended Error which covers any problems with Crypto cell (CC312) IP.

Verbosity

Using the -v option will increase the level of verbosity. It will show the ISP protocol packets for transmit and receive.



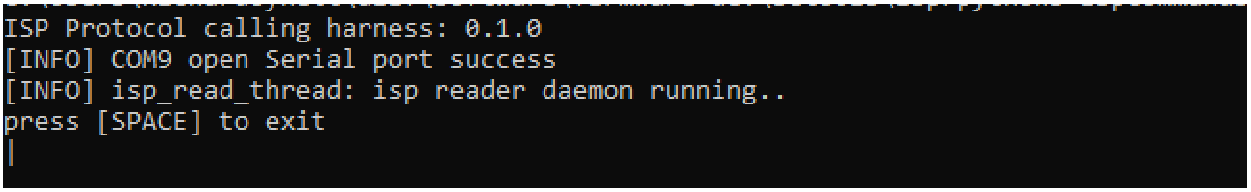
```
$ python3 ispcommands.py -c debug_enable -v
ISP Protocol calling harness: 0.1.0
[INFO] COM9 open Serial port success
[INFO] isp_read_thread: isp reader daemon running..
TX-->  length=   3 command= COMMAND_START_ISP      chksum= 0xfd
RX<--  length=   4 command= COMMAND_NAK            chksum= 0x7 error= ISP_UNEXPECTED_COMMAND
TX-->  length=   3 command= COMMAND_SECURE_DEBUG   chksum= 0xef
RX<--  length=  24 command= COMMAND_PRINT_DATA     chksum= 0x43
 secure debug enabled
TX-->  length=   3 command= COMMAND_STOP_ISP       chksum= 0xfc
RX<--  length=   3 command= COMMAND_ACK            chksum= 0xff
RX<--  length=   3 command= COMMAND_ACK            chksum= 0xff
RX<--  length=   3 command= >> COMMAND_UNKNOWN <<  chksum= 0x3f
RX<--  length=   3 command= >> COMMAND_UNKNOWN <<  chksum= 0x6c
251RX<--  length=   3 command= COMMAND_BURN_MRAM      chksum= 0x0
RX<--  length=   3 command= >> COMMAND_UNKNOWN <<  chksum= 0x3f
RX<--  length=   3 command= >> COMMAND_UNKNOWN <<  chksum= 0x6c
251RX<--  length=   3 command= COMMAND_BURN_MRAM      chksum= 0x0
RX<--  length=   3 command= >> COMMAND_UNKNOWN <<  chksum= 0x3f
RX<--  length=   3 command= >> COMMAND_UNKNOWN <<  chksum= 0x6c
251RX<--  length=   3 command= COMMAND_BURN_MRAM      chksum= 0x0
RX<--  length=   0[INFO] isp_read_thread:  ended
[INFO] COM9 closeSerial success
```

You will note that there are COMMAND_UNKNOWN messages printed. This is because not all the print traffic from the Target is using the ISP Protocol and so are interpreted as ISP packets when they are not, hence the COMMAND_UNKNOWN.

Note if you are doing MRAM updates with verbose mode enabled there will be a lot of print traffic.

Terminal

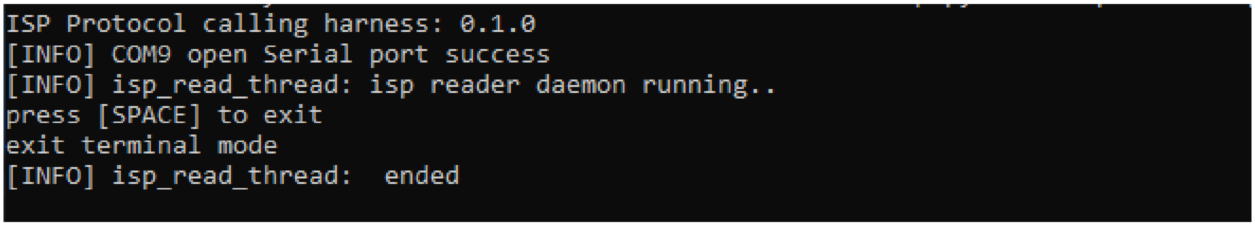
Using the -t option enters a terminal mode. This simply consumes data from the target and prints it.



```
ISP Protocol calling harness: 0.1.0
[INFO] COM9 open Serial port success
[INFO] isp_read_thread: isp reader daemon running..
press [SPACE] to exit
|
```

This is printing the flicker from SERAM.

To exit, press the [SPACE] bar:



```
ISP Protocol calling harness: 0.1.0
[INFO] COM9 open Serial port success
[INFO] isp_read_thread: isp reader daemon running..
press [SPACE] to exit
exit terminal mode
[INFO] isp_read_thread:  ended
```

Document History

Version	Date	Author	Change Log
1.0	April 1, 2021	S. SCAGLIA	Initial concept and realization
2.0	April 30, 2021	S. SCAGLIA	Added compression support. Updated write_image.py tool to support erase command and reboot device.
2.1	May 4, 2021	S. SCAGLIA	Changed attribute for signed/unsigned images.
2.2	May 21, 2021	S. SCAGLIA	Initial support for ISP is included. Updated parameters validation for app-write-mram.py
2.3	June 3, 2021	R. ONYETT	Isp support updated
2.4	June 15, 2021	R. ONYETT	Isp command new features
2.5	June 28, 2021	R. ONYETT	Updates for ISP support in standard tools
2.6	July 10, 2021	S. SCAGLIA	Added DEFERRED flag
2.7	July 30, 2021	S. SCAGLIA	Added Global Configuration
2.8	August 13, 2021	S. SCAGLIA	OEM renamed as APP
2.9	September 10, 2021	S. SCAGLIA	ISP updates
3.0	September 2021	R. ONYETT	ISP updates – get_toc, get_revision update
3.1	October 2021	S. SCAGLIA	Minor changes and review
3.2	October 2021	R. ONYETT	Updated isp mode parameters
3.3	November 2021	R. ONYETT	Updates for REV_A1, SERAM recovery
3.4	December 2021	S. SCAGLIA	Removed JTAG access
3.5	January 2022	R. ONYETT	Adding ChangeSets

Version	Date	Change Log
0.37.0	December 2021	Web release for 0.37.0 tools
0.40.0	January 2022	Updates for several commands