SATK User Guide

Table of Contents

Notices....1
Introduction....1
  Using a Sample....2
  Building and Using the Sample Yourself....3
    Scripted Processes....3
    Typed Processes....4
  SATK (Bare-Metal) Programs....4
Program 1 – Hello World....7
  Step 1 – Assemble the Program....9
  Step 2 – Run the Bare-Metal Program....10
    Configuration File....10
    Run Control File....11
    List-Directed IPL Directory....11

Notices

"Python" is a registered trademark of the Python® Software Foundation.

Introduction

READ THIS INTRODUCTION!!!

Stand Alone Tool Kit, SATK, has a number of related features. SATK consists of a set of tools for the creation of bare-metal programs targeting the mainframe environment. While the most prominent might be its assembler, ASMA, a number of other tools are provided to assist in the bare-metal program's creation and use. It is targeted primarily for use with the Hercules mainframe hardware emulator, but is not restricted exclusively to it. Other emulators and virtual environments exist.

The tools provided by SATK are platform independent, exclusively using Python for their implementation. The author uses Linux to build, test and document SATK. As a consequence, this document, with its platform specific execution examples, is Linux oriented. My apologies to the Windows community. It is the author's hope, that a new Windows user can adjust. Contributions for use of SATK with Windows are welcomed.

SATK User Guide

This document is intended to give the new user of SATK help in effectively using it through example programs as well as where within the SATK document set additional information may be found. The programs as well as tool output is provided in the `samples/guide` SATK directory. Each program is numbered and is provided with its corresponding sample and output within `samples/guide/pgm`***x***, where x corresponds to the program's number in this document.

Hercules, the primary mainframe environment targeted by SATK, requires a configuration file to run. The configuration uses a filename suffix of `.conf`. After Hercules is invoked, a run control file, suffixed with `.rc`, can execute Hercules commands, as well as host platform commands via the Hercules `sh` command. These are placed in the `samples/guide/pgm`***x*** directory. Each `pgm`***x*** directory is self contained.

To help or provide additional information on using the tools provided by SATK, each directory will contain a Bash script, `run`, that does what is needed to execute the samples. Windows users will need to revert to typed command lines or create their own `.bat` file, or other scripting language. Hopefully the `run` script will be simple enough that a Windows user can adapt it. Simply change the current working directory to `samples/guide/pgm`***x*** and execute the Bash script, `run`.

Assembler listings are too large to readily be placed in a document. To access the listing, a text editor will be required, such as Kwrite on Linux, or Wordpad of Windows. If available Visual Studio on Windows will also work. You never need to worry about line-terminations with SATK software. Python takes care of them.

Using a Sample

To help or provide additional information on using the tools provided by SATK, each directory will contain a Bash script, `run`, that does what is needed to execute the samples. Simply change the current working directory to `samples/guide/pgm`***x*** and execute the Bash script.

Executing the `run` script on Linux has these options:

`./run asm` – Assembles the sample program, `pgm`***x***`.asm`

`./run med` – Creates the sample's IPL medium (if needed)

`./run ipl` – Executes the sample with Hercules

Output files and some input files will require fully qualified filenames to be found or written. Where this is the case, a tool's `--help` option identifies a `FILEPATH` as being required for the command-line argument.

The `run` script is used by the author to construct and test the sample. It is also intended to be an aid for you to run the sample as well in your own environment.

Building and Using the Sample Yourself

It is recommended that you build the sample yourself in your own environment.

You will need the following files from the SATK sample program:

- `pgm`***x***`.asm` – the assembler source of the sample program
- `pgm`***x***`.conf` – the configuration file to run the sample with Hercules
- `pgm`***x***`.rc` – the Hercules run control file that will launch the bare metal program

You will need to determine how you will execute `tools/asma.py`, `tools/iplasma.py`, if needed, and Hercules itself. Changing the working directory to the one containing the files from SATK may simplify your execution of the various tools. To facilitate this additional Bash scripts are provided, one for each process:

- `asm` – Assembles the sample programs
- `med` – Creates the IPL medium (when used)
- `ipl` – Executes the sample bare-metal program in Hercules

It is also the goal to make migration to a different scripting language easier. These scripts are much simpler than the run script. Your directory will require a copy of `asm`, `med`, and `ipl` or altered for your local environment and choice of scripting language.

Scripted Processes

One strategy for executing a process with multiple steps is for the processes to be scripted in one or more scripts. With this strategy the typical way the script or scripts are executed is to changed the working directory to the location of the script and execute it from where it resides. Within the script, each tool and input file is specified in the script frequently with fully qualified paths as needed. The user can simply type the name of the script with a few options to execute a step in the process. This is the strategy typically used by the author.

In this approach no assumptions are made about where various components of the process reside within the file system. The previously described run script is designed with this concept. It could be moved to a different directory and run unchanged. With a few changes files and tools used by the script can be moved.

The run script within the program specific directory follows this philosophy.

Typed Processes

Another strategy is to type at a command line each step of the process. Obviously, reduction in typing is desired for multiple reasons. This tends to drive the user to setting the current working directory to one that contains each input file and in which is placed each output file.

While still scripted, the `asm`, `med`, and `ipl` Bash scripts follow this approach. They assume tool execution occurs from within the program directory and minimally make use of script variables. This allows the script to be moved to a different directory (with local changes) to another directory and executed there with minimal change.

Feedback from Windows users would be appreciated on these various approaches.

SATK (Bare-Metal) Programs

All programs created by SATK are bare-metal programs. They execute without need of an operating system, interacting directly with the hardware environment, emulated or otherwise. As such there are at most three steps as described in this table.

Step	Required or Optional	Tool	Description
1	required	`asma.py`	Assemble a bare-metal program from source
2	optional	`iplasma.py`	Create IPL capable media from assembled program
3	required	`hercules`	Test the program by performing initial program load

`asma.py` and `iplasma.py` are provided by SATK in the `tools` directory.

This document assumes three things (from the previous table):

1. **Python®** is installed and running on the host platform. SATK requires **Python**. **Python** is available from https://www.python.org/downloads/. Generally a Windows

installer is available from this site. On Linux, the operating system distributor usually has one in its package management system, if not already installed by the distributor.

2. SATK has been installed from github and ASMA runs. SATK is available at https://github.com/s390guy/SATK. The file `doc/asma/ASMA.pdf` provided with SATK describes getting started by verifying ASMA can be used. If ASMA, the assembler, runs, the other tools are also available. Building is not required.
3. The Hercules emulator has also been installed on your host platform. The latest version can be found at https://github.com/SDL-Hercules-390/hyperion. Pre-built Windows binaries can be found at http://www.softdevlabs.com/hyperion.html#prebuilt.

   Instructions for building Hercules from source can be found in the github repository.

The development of bare-metal programs is an esoteric area of knowledge. This is true for mainframe systems. Any program that interacts directly with the hardware, that is, without an operating system, is a bare-metal program. An operating system, by this definition, is a bare-metal program. For the purposes of this document, "hardware" may be physical hardware or emulated hardware.

The distinguishing characteristic for bare-metal programs, at least in this document, is *how they are placed in memory*. To better describe this characteristic, three terms are used here to differentiate bare-metal programs:

- **IPL Program** – A bare-metal program *loaded into memory by the mainframe hardware* NOT requiring a boot loader program.
- **Boot Loader Program** – A bare-metal program that itself is loaded directly into memory as an IPL program by the hardware, but is *distinguished by providing the service of loading at least one other program from a boot medium into memory and transferring control to the loaded program*. A boot loader, by this definition, could provide other services, making them available to a loaded program.
- **Booted Program** – A bare-metal program that interacts directly with hardware but is *loaded into memory by a boot loader* from which control is passed.

Hercules supports a form of IPL program that does not require a typical external medium for the purpose of the IPL, for example a disk volume. This is called list-directed IPL. ASMA supports creating this as output from the assembler directly. Mainframes introduced this form of IPL when Linux started to be supported and installed from the host management console. Hercules uses a directory for this purpose. ASMA can create this directory with the required content. This is why some samples do not require the use of `./run med`.

Key to the nature of bare-metal programs is from the above definition: *Any program that interacts directly with the hardware … is a bare-metal program*. For mainframes, and most other systems, that means the bare-metal program operates in privilege program state. This is managed by the mainframe’s PSW. The IPL PSW introduced to the CPU to start program execution must indicate privilege program state. When a bare-metal program starts execution it does so in real storage.

The bare-metal program must deal with the possible presence of interruptions, the most important being the program class of interruptions. Failing to do so will result in the CPU entering an interruption loop. The only way to get out of this is to manually stop the CPU. Stopping the CPU is done by use of the Hercules console command:

```
stopall
```

Understanding how to interact directly with the hardware requires understanding of how to exactly do that. Depending upon the architecture for which you are targeting your program, the appropriate *Principles of Operations* manual is strongly recommended. Most all of the details you will require are explained in this manual. Where external devices are involved, various other manuals will benefit for the specific device. For the earlier systems, there is a large amount of documents at www.bitsavers.org. These are largely S/360 and S/370 era documents, although some later documents are available. Much more recent documents can be located at the mainframe’s manufacturer’s websites.

Assigned storage areas used by the hardware and sometimes SATK must be honored.

Program 1 – Hello World

Program 1 is the legendary “Hello World” program. It does not require use of any I/O because it uses a DIAGNOSE instruction to deliver the message, part of Hercules VM emulation.

Program: `samples/guide/pgm1/pgm1.asm`

Hardware Architecture: S/370

Bare-Metal Program Type: IPL Program

IPL: list-directed

Macros: None

Program Description:

```
         TITLE 'PGM1 - HELLO WORLD'
```
Gives the assembly listing a title.

```
         PRINT DATA          See all object data in listing
```
See all of the data generated by the assembler.

```
PSWSECT  START 0,IPLPSW      Start the first region for the IPL PSW
* This results in IPLPSW.bin being created in the list directed IPL directory
         PSWEC 0,0,0,0,PGMSECT The IPL PSW for this program
```
As with all bare-metal programs, it requires an IPL PSW to be placed in absolute addresses 0-7. To accomplish this a memory region is started, named `IPLPSW`, at address 0. The PSW reflects privilege mode and the address of the program itself, `PGMSECT`. In addition to starting a new memory region, it also starts a named control section. In the list-directed IPL control file, this area of memory becomes a binary image file named `IPLPSW.bin`. Both the second operand of the START operation (starting the memory region) and the PSWEC operation (generating a PSW in the same manner as a CCW) are ASMA specific.

```
PGMSECT  START X'300',IPLPGM1  Start a second region for the program itself
```
The IPL program is placed in a separate memory region, `IPLPGM1`. A second START operation is used for this. Unlike legacy assemblers, ASMA, accepts multiple START statements. The new control section, `PGMSECT`, has its own location counter set to absolute address X’300’. X’300’ is selected to avoid hardware assigned storage areas and the SATK reserved areas in X’200’-X’2FF’. As with the initial START operation, the second one will create another file in the list-directed IPL directory, `IPLPGM1.bin`.

Both files created within the list-directed IPL directory are created from the control sections which make up each memory region. In addition, each file will be referenced, with its respective starting location, in the list-directed IPL control file within the same directory. The ASMA `-g` command line option specifies the path and file name of the control file. The directory for each of the binary files and control file is inferred from the `-g` argument. The three files are placed in the same directory.

```
          BALR  12,0            Establish my base register
          USING *,12            Tell the assembler
```

Usual program set up for its intended base register.

```
          MVC   X'68'(8,0),PGMEX  Set a NEW PSW to trap program interruptions
PGMEX     PSWEC 0,0,2,0,X'28'   Points to the program OLD PSW
```

This is the earliest point in the program that a protection against an uninterrupted program interruption loop can be established. Why? Because the MVC instruction requires a base register. Once the base register is established, the program can place in the assigned storage area used by the hardware the program interruption New PSW. This PSW gets loaded in case a program interruption occurs. It stops the CPU automatically by have the wait bit set, the 2 in the third operand. All of the maskable interrupts are disabled. The combination of the two conditions effectively stops the CPU. Hercules will display such a PSW on the console, providing an opportunity to communicate why the program came to a halt. By using X'28' as the address, the PSW indicates that it was a program interruption and the Old PSW (where the condition occurred) is at real address X'28'.

```
          LM    8,10,DIAGPRMS   Load the DIAGNOSE instruction's registers
          DIAG  8,10,X'008'     Issue the Hello World message to the console
DIAGPRMS  DC    A(COMMAND),A(0),X'00',AL3(CMDLEN),A(0)
COMMAND   DC    C'MSG * '                    VM emulated command
MESSAGE   DC    C'Hello Bare-Metal World!'   Message text sent to self
CMDLEN    EQU   *-COMMAND
```

This sequence initializes the registers used by the DIAGNOSE instruction, from `DIAGPRMS`. In turn, the DC's point to the message and contain its length, along with the flag byte.

```
          BNZ   BADCMD   If a non-zero condition code set, the DIAGNOSE failed
          LTR   10,10    Was a non-zero return code set by the DIAGNOSE?
          BNZ   BADCMD   ..Yes, abnormal program termination
```

The results of the DIAGNOSE are checked here, first, the condition code and then the return code. If either fails the program branches to the label `BADCMD`. Instruction execution simply falls through if the DIAGNOSE was successful.

```
          LPSW  GOOD       ..No, normal program termination
          SPACE 1
BADCMD    LPSW  BAD        End with a bad address in PSW field
GOOD      PSWEC 0,0,2,0,0       Successful execution of the program
BAD       PSWEC 0,0,2,0,X'DEAD' Unsuccessful execution of the program
```

Success or failure is indicated by which PSW is explicitly loaded by the program. In both cases a disabled wait state is established as was the case with the established New PSW. The difference is in the address field. For a successful program execution, an address of 0 is used. For an abnormal termination, an address of DEAD, all hexadecimal digits, is used. The LPSW instruction loads the PSW from memory into the active PSW.

So, the instruction address of the final program PSW indicates how the program was executed as in this table.

PSW Address	Meaning
`X’0000’`	Successful execution of the program
`X’0028’`	An unexpected program interruption occurred
`X’DEAD’`	An abnormal program termination occurred

Step 1 – Assemble the Program

Change the working directory to the directory into which you have moved the files from SATK.

Adjust the copied `asm` script with local environment changes (and adjust to your scripting language if needed).

The assembly is invoked with this command in the `asm` script:

```
${ASMA} -t s370 -d --stats -g ldipl/pgm1.txt -l asma-$sfx.txt pgm1.asm
```

The script variable `${ASMA}` points to the local location of the `asma.py` script within SATK’s `tools` directory.

`asma.py` is presented with a number of command-line arguments. These affect the assembly in the following ways.

`-t s370` targets the S/370 architecture by the assembler. It restricts recognized instruction mnemonics to those recognized by systems implementing S/370.

`-d` influences the content of the listing. It causes ASMA to display in hexadecimal and characters the binary content it has created.

`--stats` causes `asma.py` to display on the console run-time time statistics and where time is spent. A statement rate is also provided.

`-g ldipl/pgm1.txt` identifies the type of output to be generated by the assembler, a list-directed IPL directory content, and where this content is to be placed. The `-d` argument prints the content of the binary files.

`-l asma-$sfx.txt` directs the assembler to generate a listing and into which file it is to be written. The use of the `$sfx` script variable allows the listing file name to include the date and time when it was created.

`pgm1.asm` identifies for the source file assembled by ASMA. It is a positional parameter and must always be the last in the command line.

Step 2 – Run the Bare-Metal Program

Change the working directory to the directory into which you have moved the files from SATK.

Execution of the program involves performing a list-directed IPL upon the `-g` file created by ASMA.

The IPL is initiated with these two commands in the `ipl` script:

```
      export HERCULES_RC=pgm1.rc
      ${HERCULES} -v -f pgm1.conf  >> log-${sfx}.txt
```

The first statement identifies to Hercules where to find the run control file, in this case `pgm1.rc`. This statement creates both a script variable and then provides it as a platform environment variable for access by Hercules. Script variables only exist within the script. Environment variables are made available to other programs.

The second statement actually launches the Hercules emulator. The `${HERCULES}` script variable identifies where on the local system the Hercules executable file is located.

The following command-line arguments influence Hercules execution.

`-v` causes Hyperion to provide verbose messages in the log.

`-f pgm1.conf` identifies to Hercules its configuration file.

The log file created by Hercules is sent to the file `log-${sfx}.txt`. As with the assembler listing, a date and time is added to the file name.

Configuration File

The Hercules configuration file has the following contents.

```
      # Hercules sample configuration file for pgm1.asm
      ARCHMODE S/370          # S/370 targeted by pgm1.asm
      MAINSIZE 16K            # Sample starts at X'300'
      NUMCPU   1
      DIAG8CMD enable         # Allow use of DIAG X'008'

      # Console Device
      000F 3215-C /           # Required by mainframe systems
```

`ARCHMODE S/370` is specified because that is the architecture targeted by the program with the assembler's -t command-line argument.

`MAINSIZE 16K` is used because this is a small program and doesn't need much to run. Additionally, 16K will probably suffice for most bare-metal programs.

`NUMCPU 1` specifies one CPU which is the norm for S/370 systems. And pgm1.asm does not require more than one CPU.

`DIAG8CMD enable` allows the DIAGNOSE function X'008' to be used by the program. In as much as this is how `pgm1.asm` conveys the Hello World message, it is required.

The single integrated console device is specified because mainframe systems require at least one console. It is available at address `000F`.

Run Control File

The run control file is run after Hercules has configured the system with the configuration file. It contains Hercules console commands. All could be entered by means of the Hercules console itself. This simply automates the process.

The run control file has these contents:

```
      # This file is intended for use with Hercules Hyperion
      # Comment this statement if you do not want to trace
      t+
      # Perform the list-directed IPL
      ipl ldipl/pgm1.txt
```

The `t+` Hercules console command causes instruction tracing to be performed. It is extremely useful during debugging of bare-metal programs, so is enabled here.

The `ipl` command performs the actual IPL, launching the bare-metal program from the list-directed IPL directory created by the ASMA `-g` argument.

List-Directed IPL Directory

The list-directed IPL directory's contents are created by the ASMA `-g` command-line argument. This directory contains three files related to the IPL function, all constructed by ASMA:

- `IPLPGM1.bin` – the bare-metal program's binary content as assembled (this is why the ASMA `-d` option is used).
- `IPLPSW.bin` – the IPL PSW used to start the execution of the program by Hercules following IPL.
- `pgm1.txt` – the IPL control file.

The IPL control file dictates where in memory the binary files are loaded, their starting memory addresses. The address was specified in the ASMA `START` operation's first parameter that initiated by binary content with its second parameter.

```
IPLPSW.bin 0x0
IPLPGM1.bin 0x300
```

When the `ipl` Hercules console command is executed by the run control file, this control file in the list-directed IPL directory controls where the file content is placed in memory.

The `IPLPSW.bin` file is placed at address X'0'. From the first 0 bytes of storage, Hercules initializes the PSW. After which, control of the CPU is passed to the loaded bare-metal program.

The Hercules console command `quit` may be entered to terminate Hercules execution.